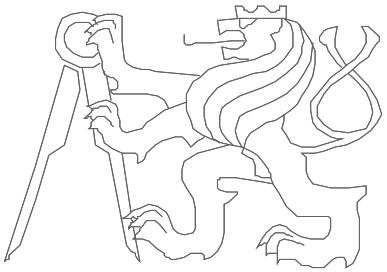
Computer Architectures

Integer Arithmetic

Richard Šusta, Pavel Píša



Czech Technical University in Prague, Faculty of Electrical Engineering

Important Introductory Note

- The goal is to understand the structure of the computer so you can make better use of its options to achieve its higher performance.
- It is also discussed interconnection of HW / SW
- **Webpages:**
  https://cw.fel.cvut.cz/wiki/courses/a0b36apo/start
  https://dcenet.felk.cvut.cz/apo/ - *they will be opened*
- Některé navazující předměty:

  B4M35PAP - Advanced Computer Architectures

  B3B38VSY - Embedded Systems

  B4M38AVS - Embedded Systems Application

  B4B35OSY - Operating Systems (OI)

  B0B35LSP – Logic Systems and Processors (KyR + part of OI)
- Prerequisite: Šusta, R.: APOLOS , CTU-FEE 2016, 51 pg.

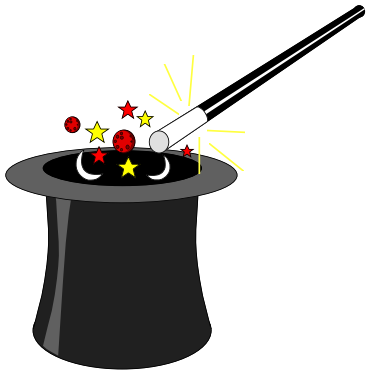
Important Introductory Note

- The course is based on a world-renowned book of authors Paterson, D., Hennessey, V.: **Computer Organization and Design**, The HW/SW Interface. Elsevier, ISBN: 978-0-12-370606-5

David Andrew Patterson
University of California, Berkeley
Works: RISC processor MIPS,
RAID, Clusters

John Leroy Hennessy
10th President of Stanford University
Works: RISC processors MIPS,
DLX a MMIX

Computers



or

where they are heading…

Budoucnost počítačů

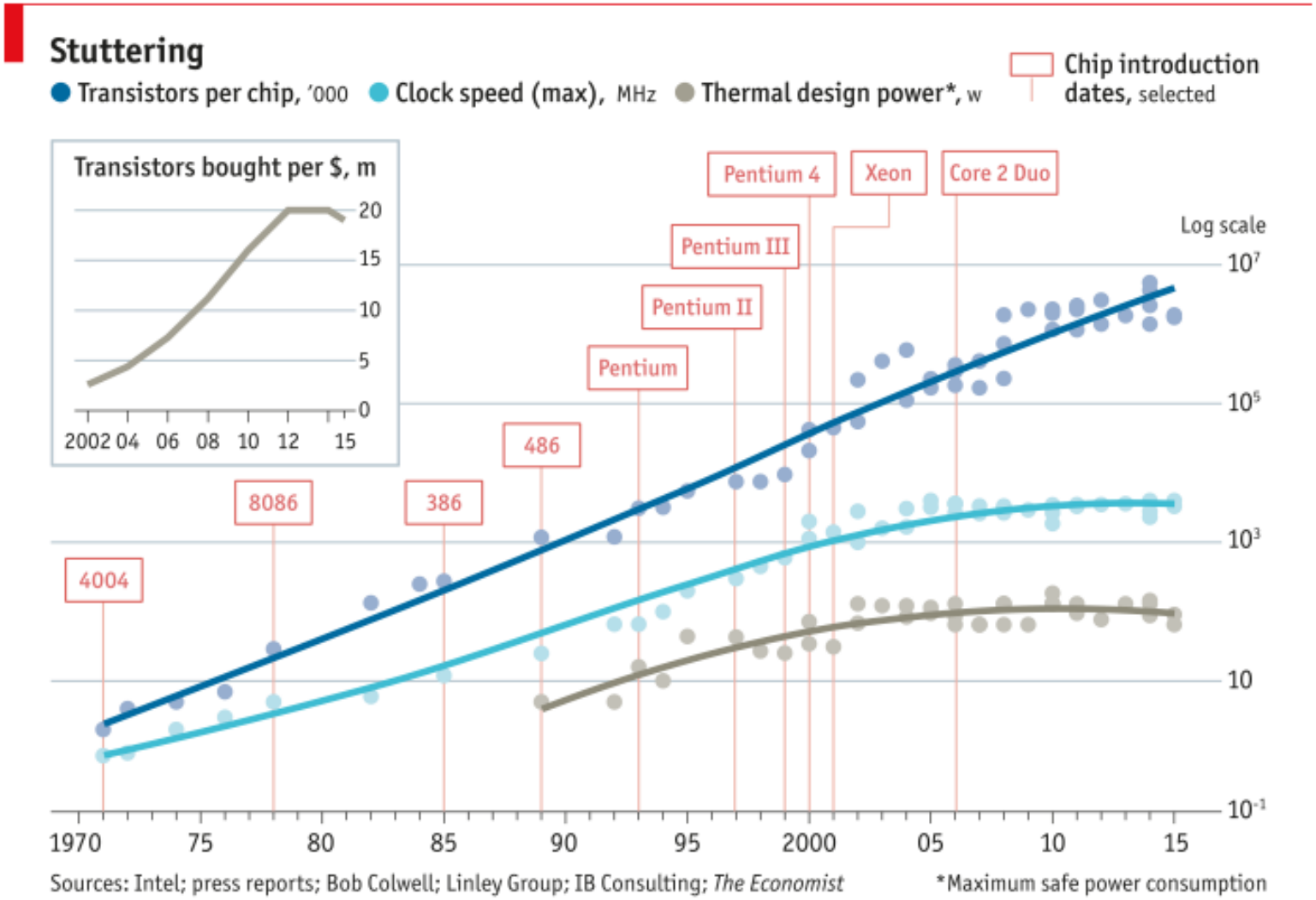
Can weather forecast be predicted?

Can the development of computer technology be predicted?

Source: https://www.hmv.com/

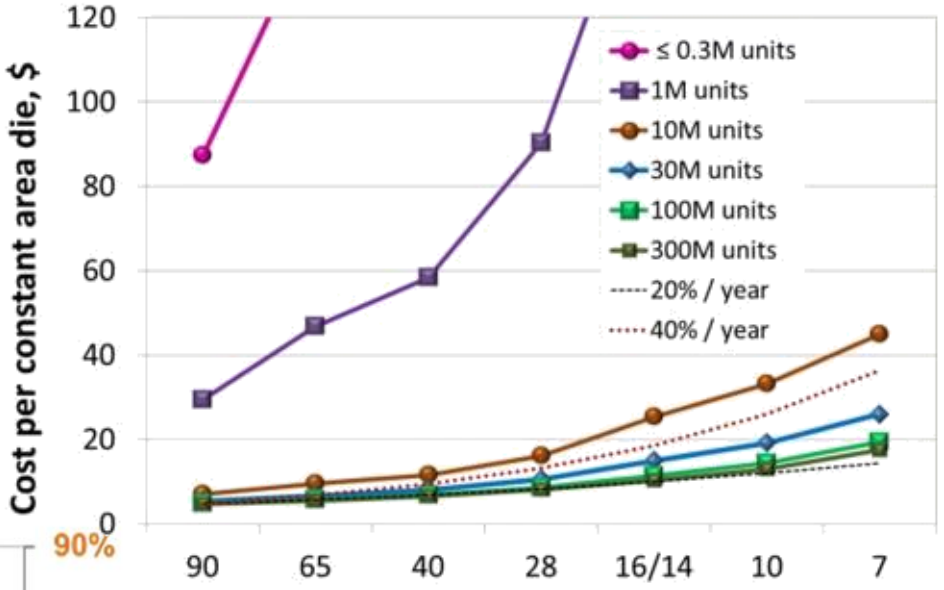
Moore's Law

Gordon Moore, founder of Intel, in 1965: " *The number of transistors on integrated circuits doubles approximately every two years* "

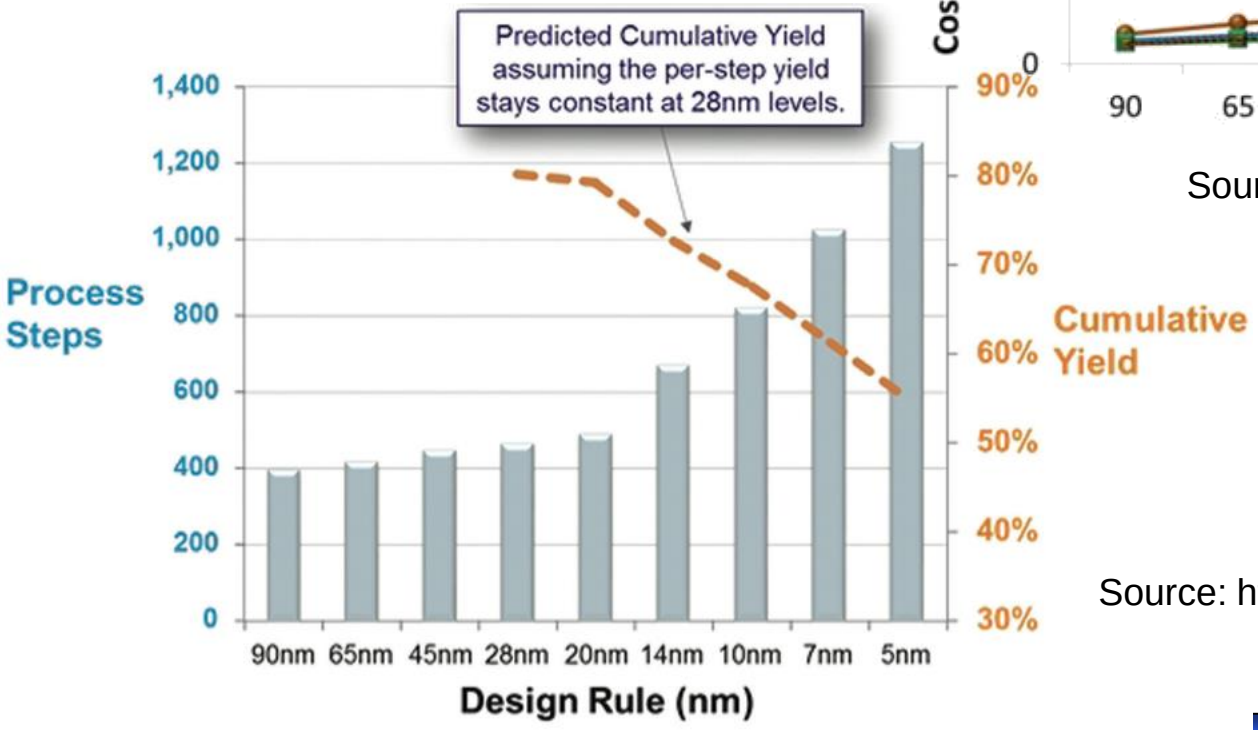


The cost of production is growing with decreasing design rule

Moore's Law will be stopped by cost…



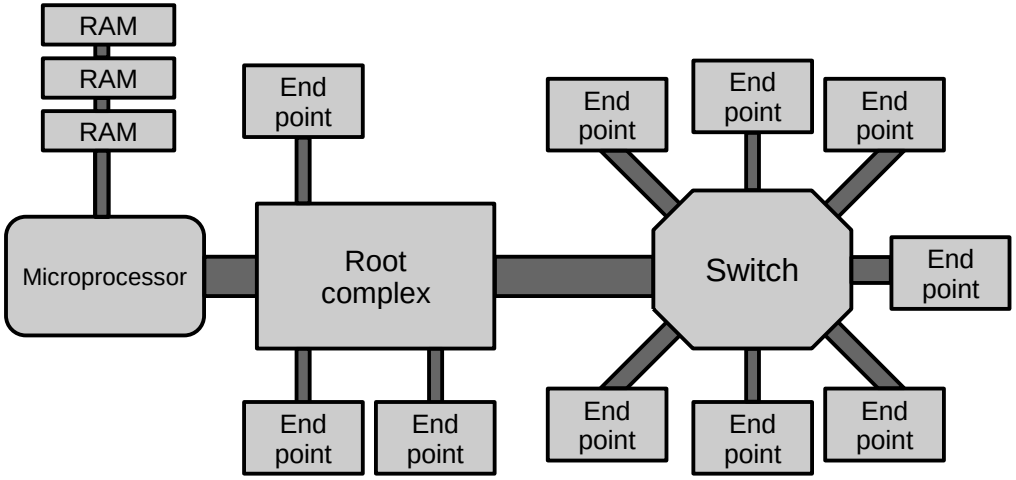
Source: http://www.eetimes.com/



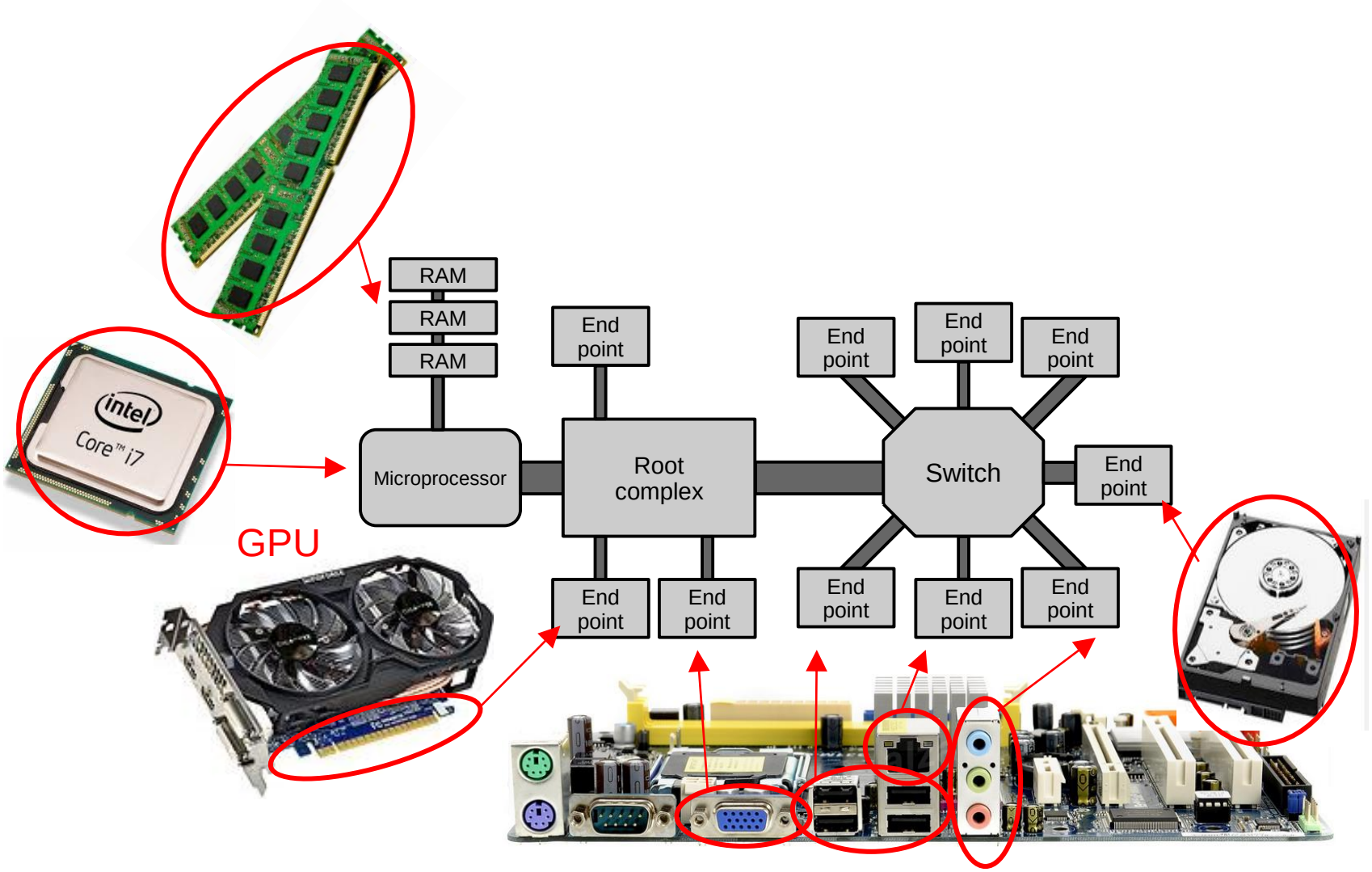
Source: http://electroiq.com/

Today PC Computer Base Platform – Motherboard

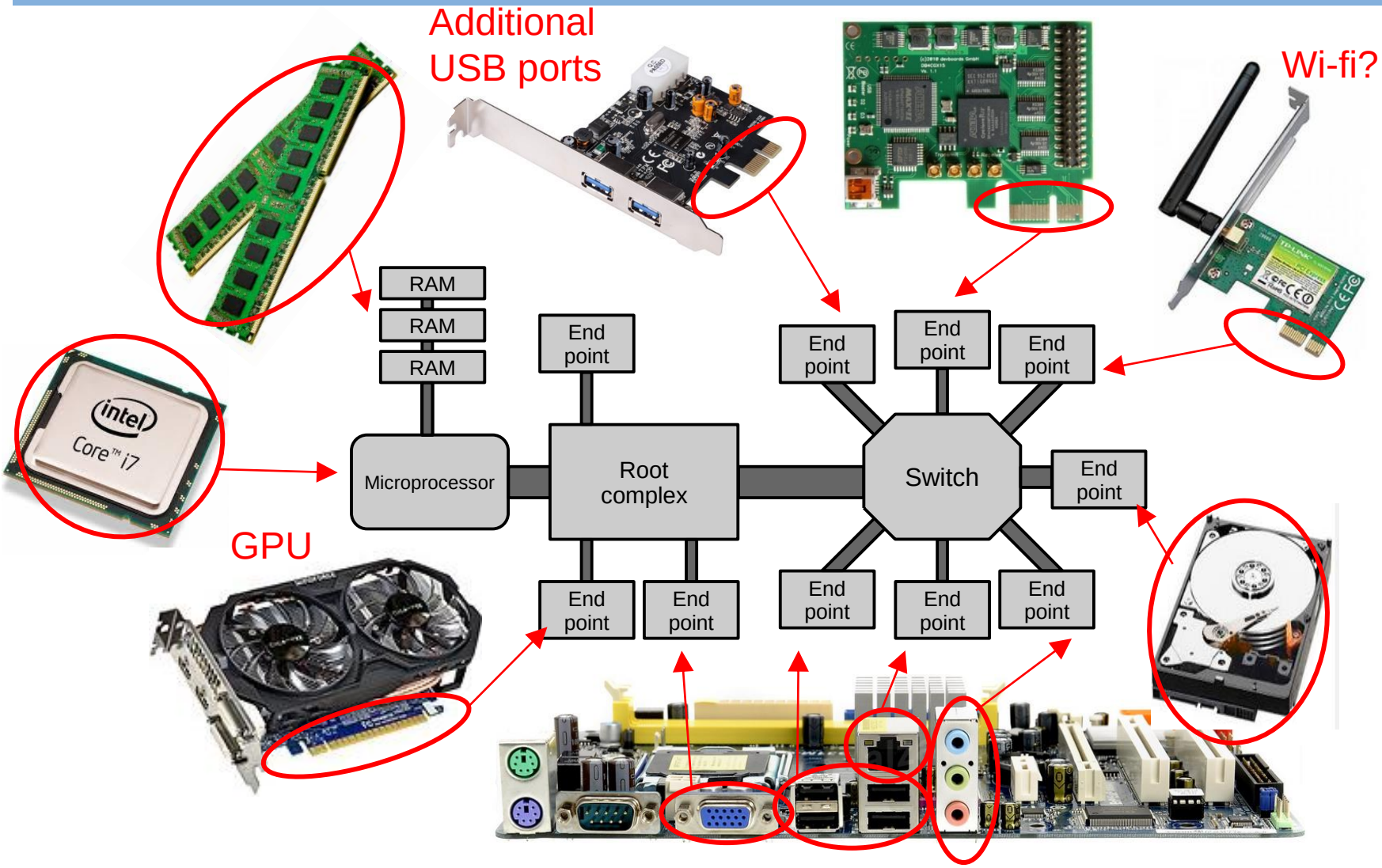
Block Diagram of Components Interconnection



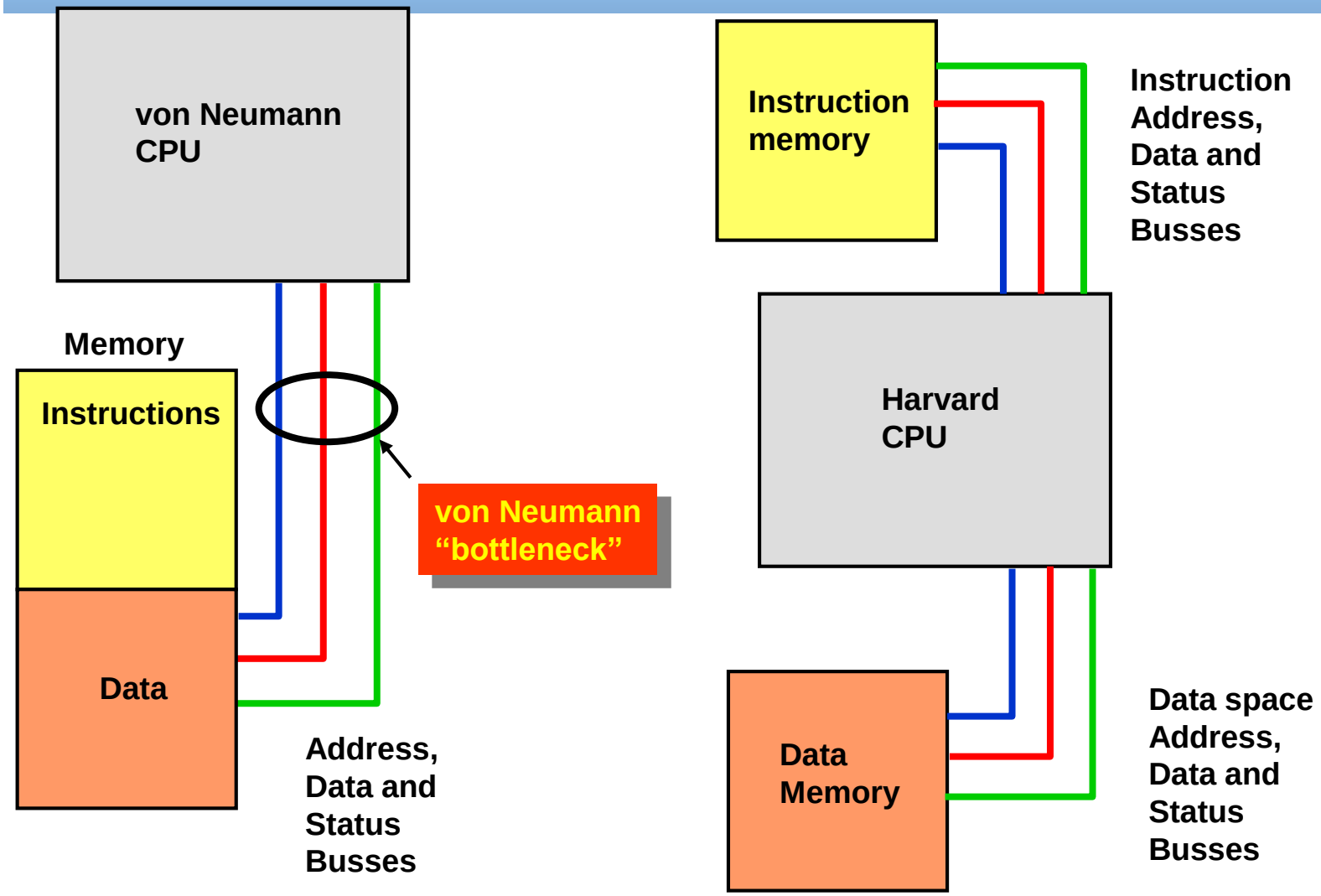
Block Diagram of Components Interconnection



Block Diagram of Components Interconnection



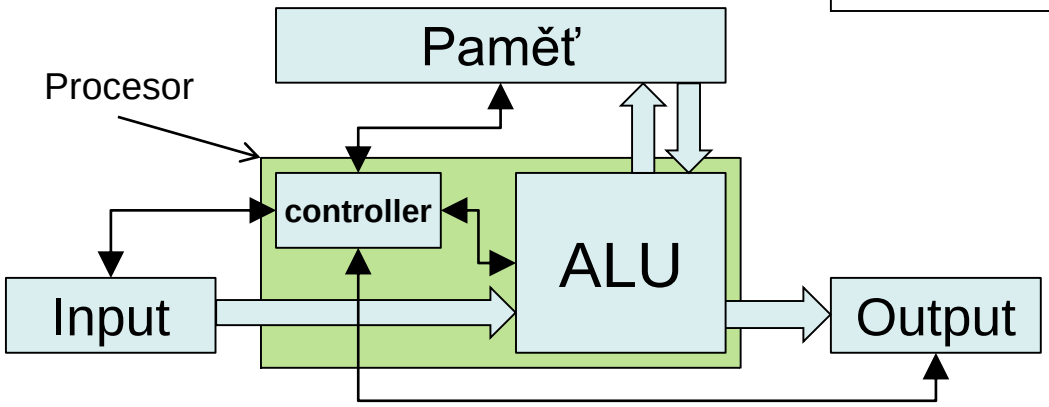
Von Neumann and Harvard Architectures



[Arnold S. Berger: Hardware Computer Organization for the Software Professional]

John von Neumann

Princeton Institute for Advanced Studies



28. 12. 1903 -
8. 2. 1957

5 units:

- A processing unit that contains an arithmetic logic unit and processor registers;
- A control unit that contains an instruction register and program counter;
- Memory that **stores data and instructions**
- External mass storage
- Input and output mechanisms

Samsung Galaxy S4 inside

- **Android 5.0 (Lollipop)**
- **2 GB RAM**
- **16 GB user RAM user**
- **1920 x 1080 display**
- **8-core CPU** (chip Exynos 5410**):**
  - **4 cores** 1.6 GHz ARM Cortex-A15
  - **4 cores** 1.2 GHz ARM Cortex-A7

Samsung Galaxy S4 inside

Source: http://www.techinsights.com/about-techinsights/overview/blog/samsung-galaxy-s4-teardown/

Samsung Galaxy S4 inside

Power management

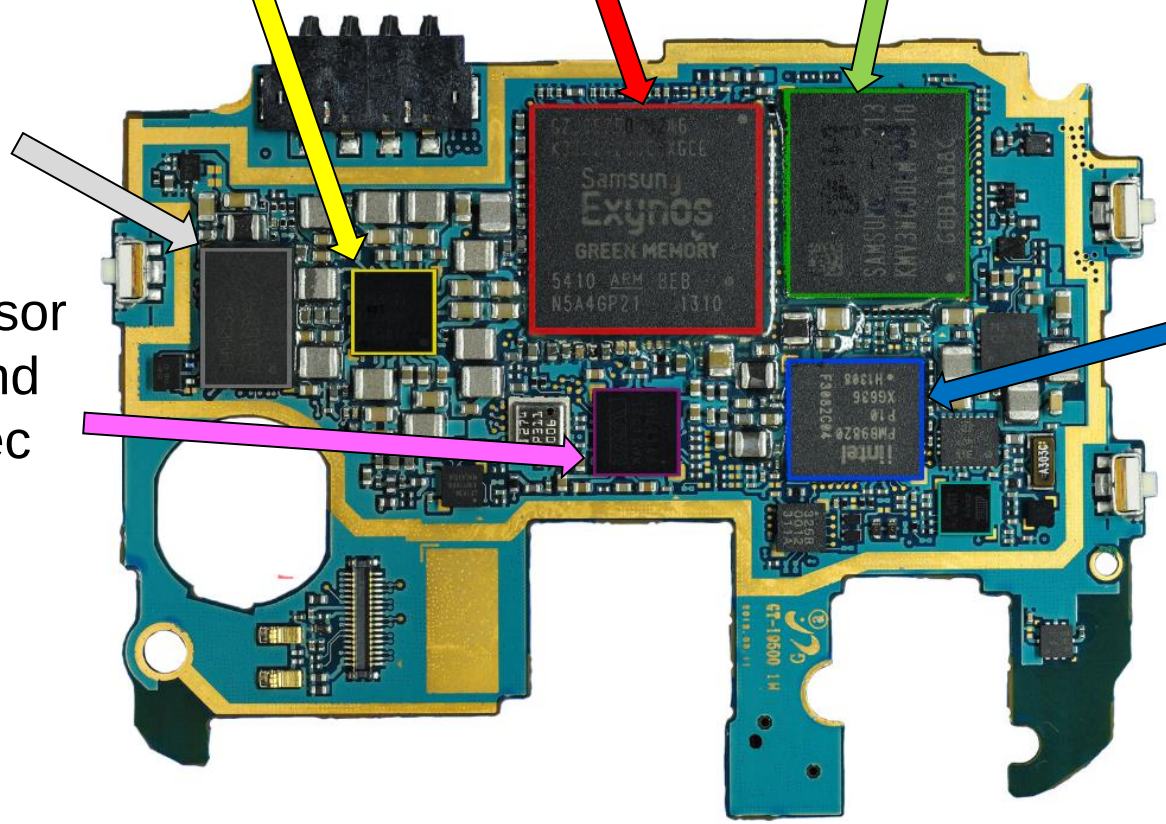
Exynos 5410 (**8-core CPU + 2GB DRAM**)

Multichip memory: 64 MB DDR SDRAM, **16GB NAND Flash**, Controller

Wi-fi (broadcom BCM4335)

DSP processor for voice and audio codec

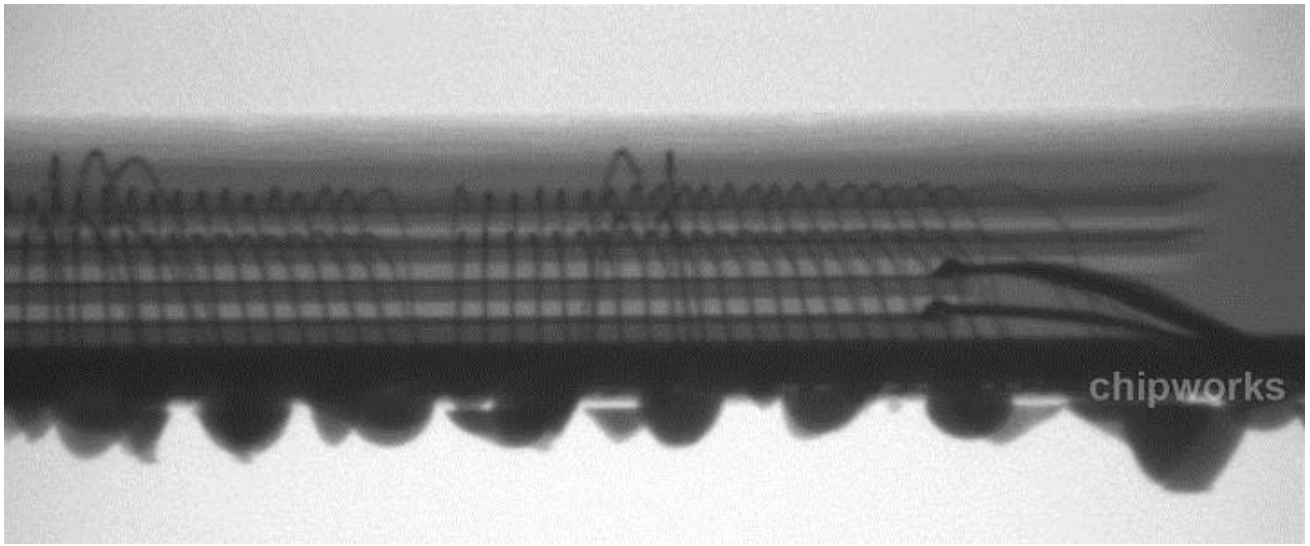
Intel PMB9820 baseband processor (functions radio - EDGE, WCDMA, HSDPA/HSUPA)



Source: http://www.techinsights.com/about-techinsights/overview/blog/samsung-galaxy-s4-teardown/

Samsung Galaxy S4 inside

X-ray image of Exynos 5410 hip from the side :



- We see that this is QDP (Quad die package)

To increase capacity, chips have multiple stacks of dies.
*A **die**, in the context of integrated circuits, is a small block of semiconducting material on which a given functional circuit is fabricated. [Wikipedia]*

Zdroj: http://gamma0burst.tistory.com/m/600

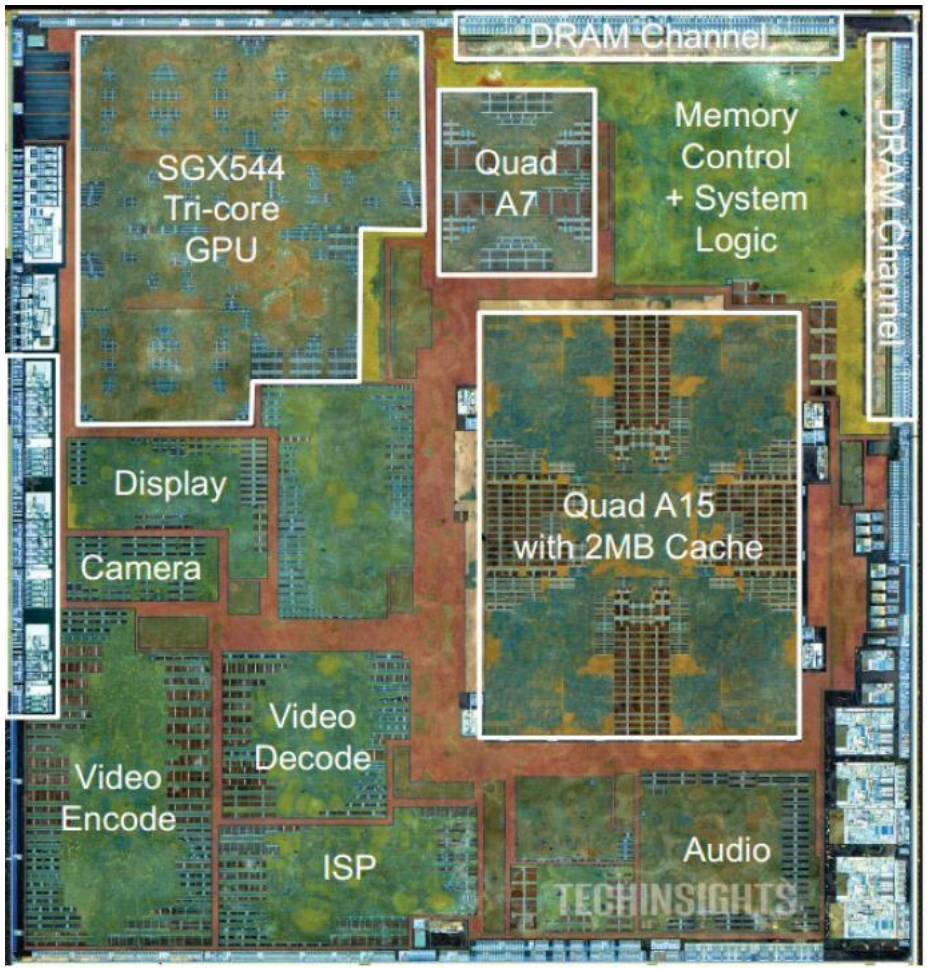
Samsung Galaxy S4 inside

Chip Exynos 5410 – here, we see DRAM

Source: http://www.embedded-vision.com/platinum-members/embedded-vision-alliance/embedded-vision-training/documents/pages/computational-photography-part-2

Samsung Galaxy S4 inside

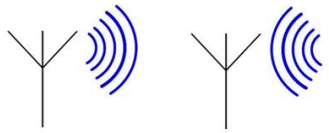
Chip Exynos 5410



- Note the different sizes of 4 cores A7 and 4 cores A15
- On the chip, other components are integrated outside the processor: the GPU, Video coder and decoder, and more. This is SoC (System on Chip)

Source: http://www.embedded-vision.com/platinum-members/embedded-vision-alliance/embedded-vision-training/documents/pages/computational-photography-part-2, http://gamma0burst.tistory.com/m/600

Samsung Galaxy S4 inside



NAND flash (16GB)

Accelerometer

GPS

Wi-fi

Baseband processor

Application processor: Exynos

Memory I/F (LPDDR3, eMMC, SD)

Peripheral I/F

CPU Cortex A15 Quad core

CPU Cortex A7 Quad core

GPU SGX544 Tri core

ISP

Camera

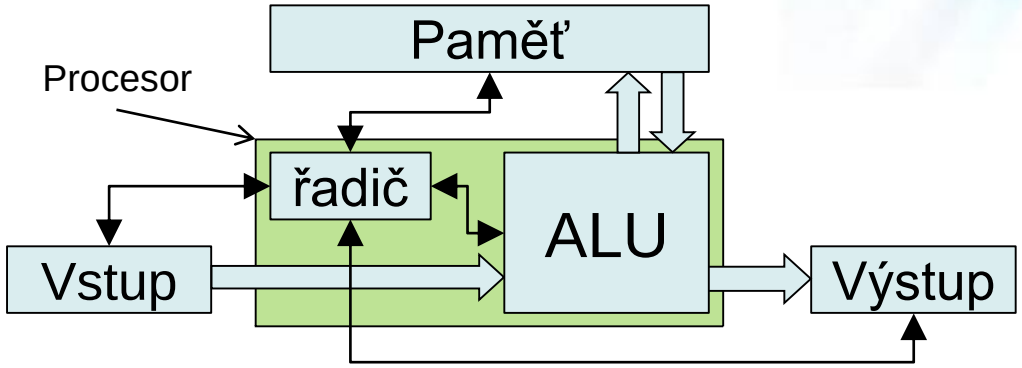
Display

High speed I/F (HSIC/ USB)

Audio

DSP processor for audio

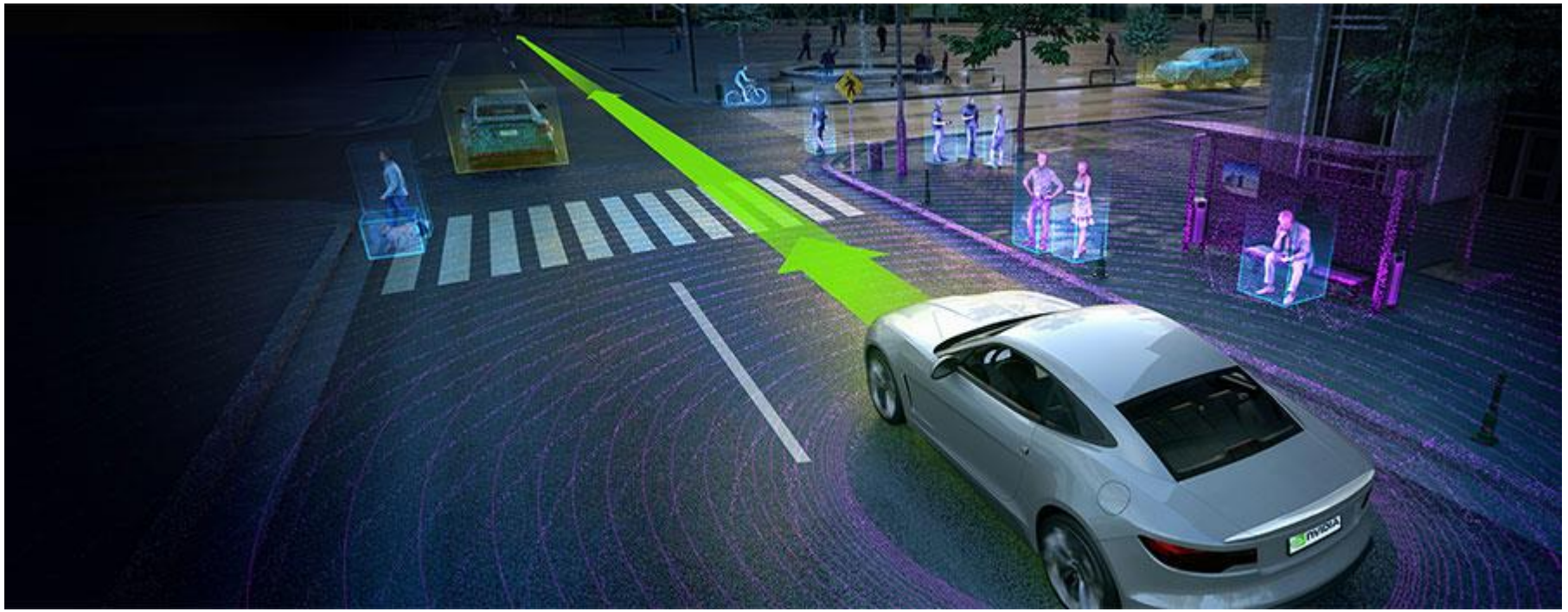
Common concept



- The processor performs stored memory (ROM, RAM) instructions to operate peripherals, to respond to external events and to process data.

Example of Optimization

Autonomous cars



Source: http://www.nvidia.com/object/autonomous-cars.html

- Many artificial intelligence tasks are based on deep neural networks (deep neural networks)

Neural network passage -> matrix multiplication

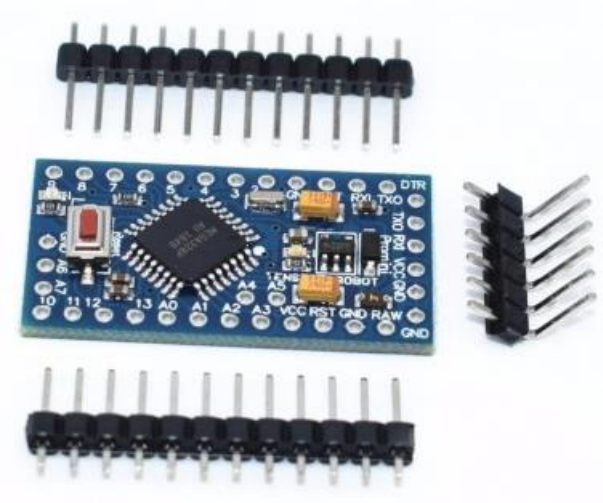
- How to increase calculation?
- The results of one of many experiments
  - Naive algorithm (3 × for) – 3.6 s = 0.28 FPS
  - Optimizing memory access – 195 ms = 5.13 FPS
    (necessary knowledge of HW)
  - 4 cores– 114 ms = 8.77 FPS
    (selection of a proper synchronization)
  - GPU (256 processors) — 25 ms = 40 FPS
    (knowledge of data transfer between CPU and coprocessors)

- Source: Naive algorithm, library Eigen (1 core), 4 cores (2 physical on i7-2520M), GPU results Joela Matějka from http://industrialinformatics.cz/

And Other Systems?

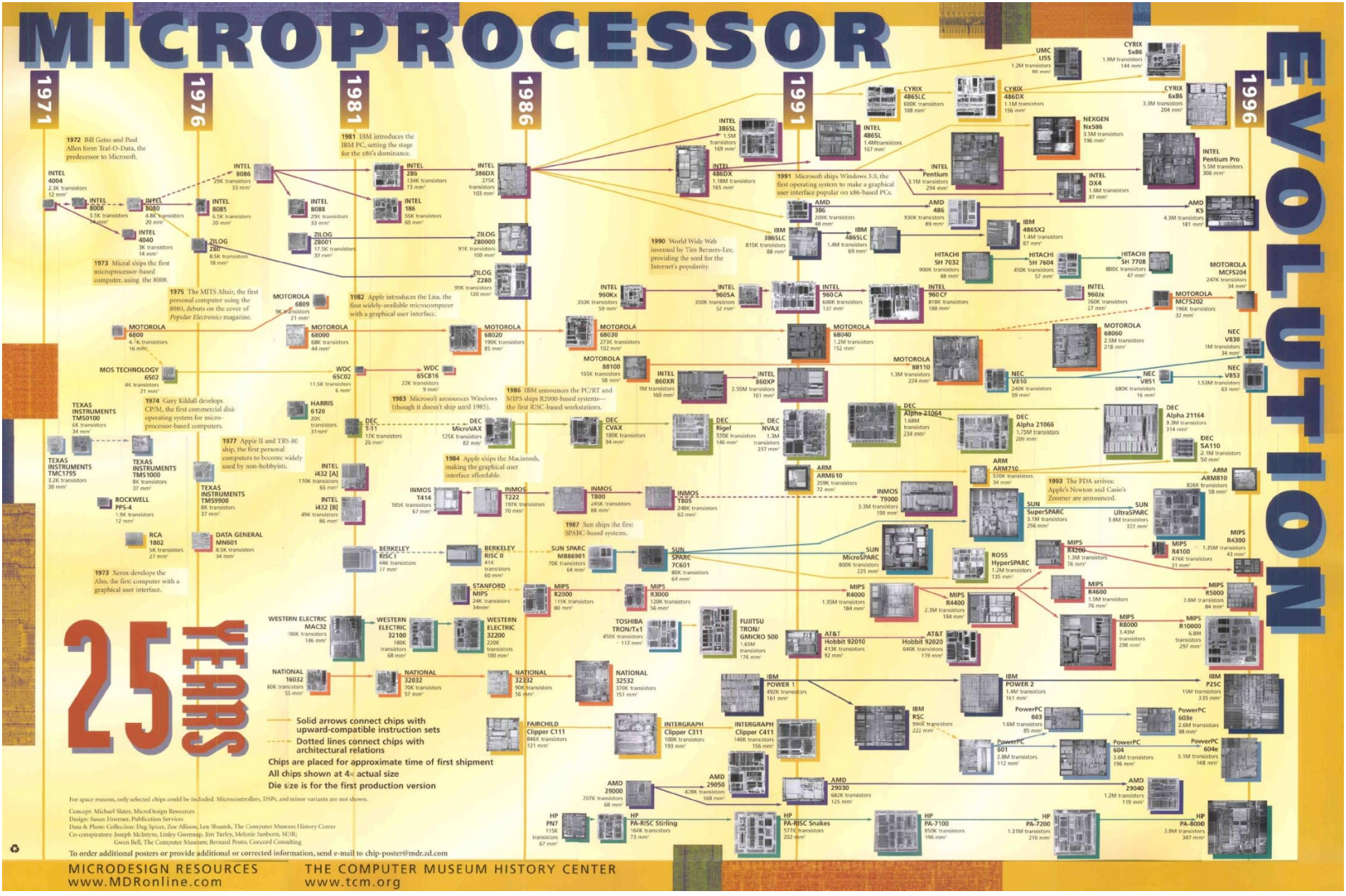
- Using GPUs, we process 40 fps.
- but cars have enough power for them...

- But in an **embedded** device, it is sometimes necessary to reduce its consumption and cost. There are used very simple processors or microcontrollers, sometimes without real number operations, and programmed with low-level C language.



There [were/are/will be] many manufactures of processors

http://research.microsoft.com/en-us/um/people/gbell/CyberMuseum_contents/Microprocessor_Evolution_Poster.jpg

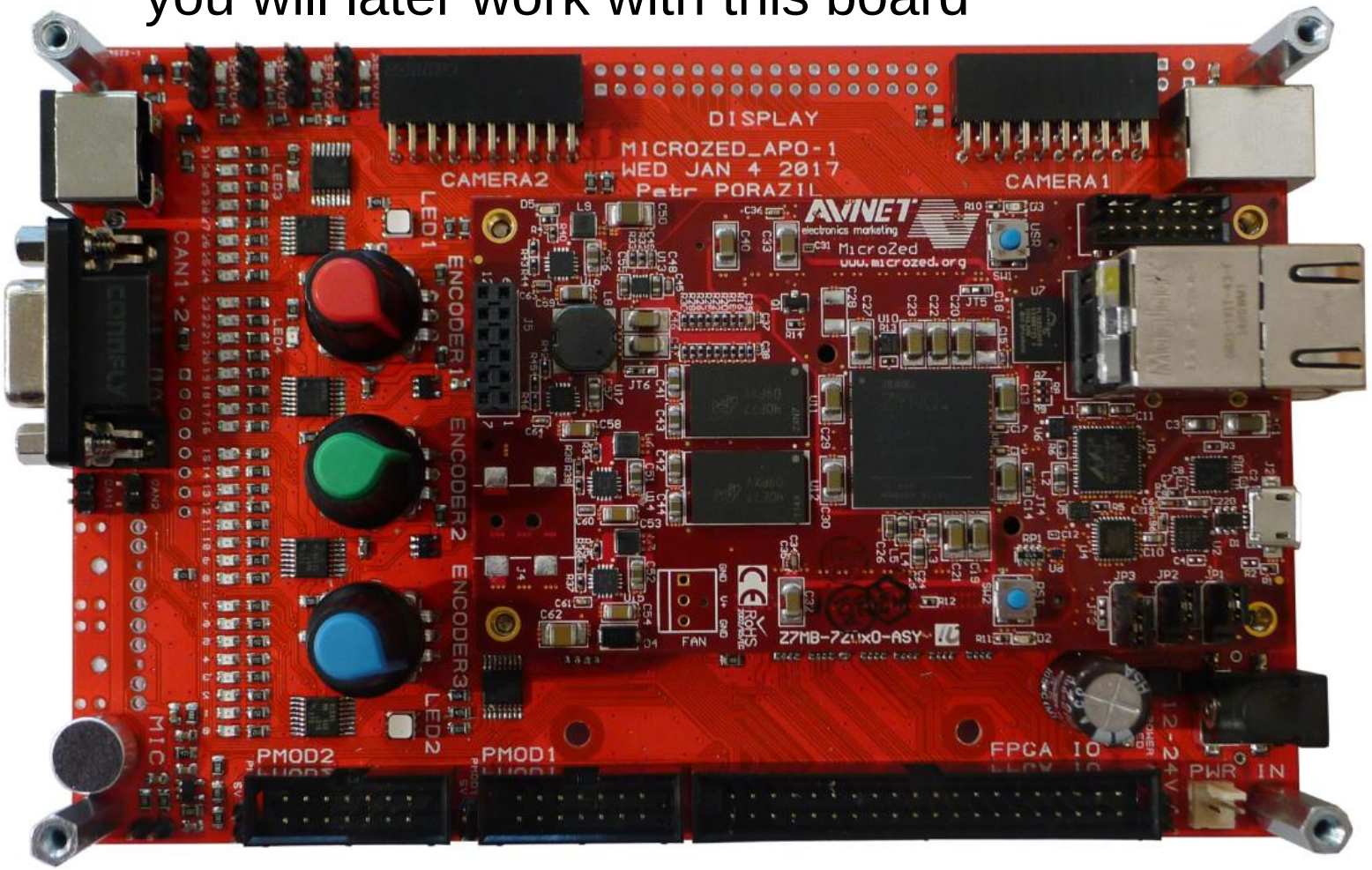


Reasons to study computer architectures

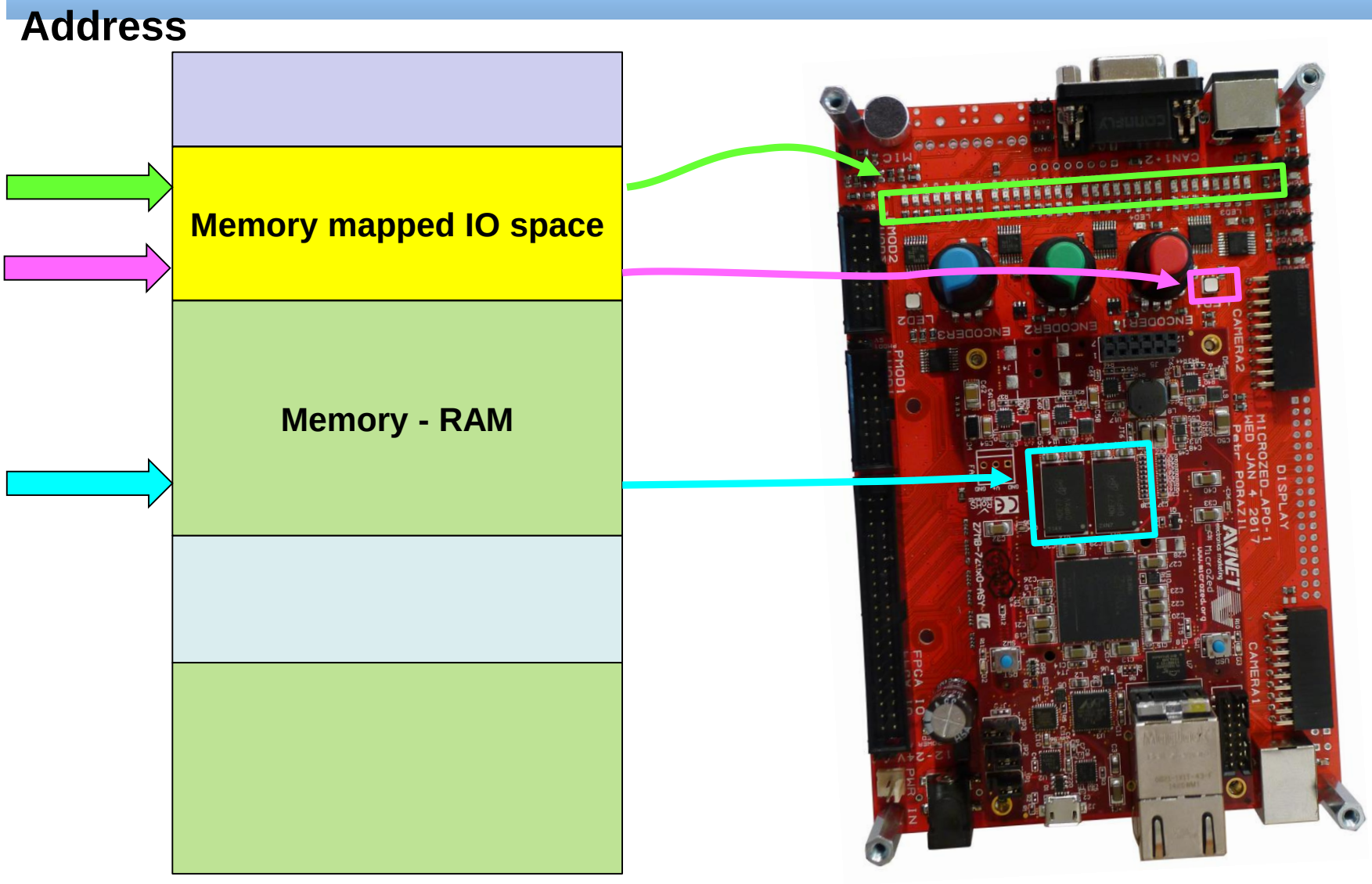
- To invent/design new computer architectures
- To be able to integrate selected architecture into silicon
- To gain knowledge required to design computer hardware/systems (big ones or embedded)
- To understand generic questions about computers, architectures and performance of various architectures
- **To understand how to use computer hardware efficiently** (i.e. how to write good software)
  - It is not possible to efficiently use resources provided by any (especially by modern) hardware without insight into their constraints, resource limits and behavior
  - It is possible to write some well paid applications without real understanding but this requires abundant resources on the hardware level. But no interesting and demanding tasks can be solved without this understanding.

Another motivation

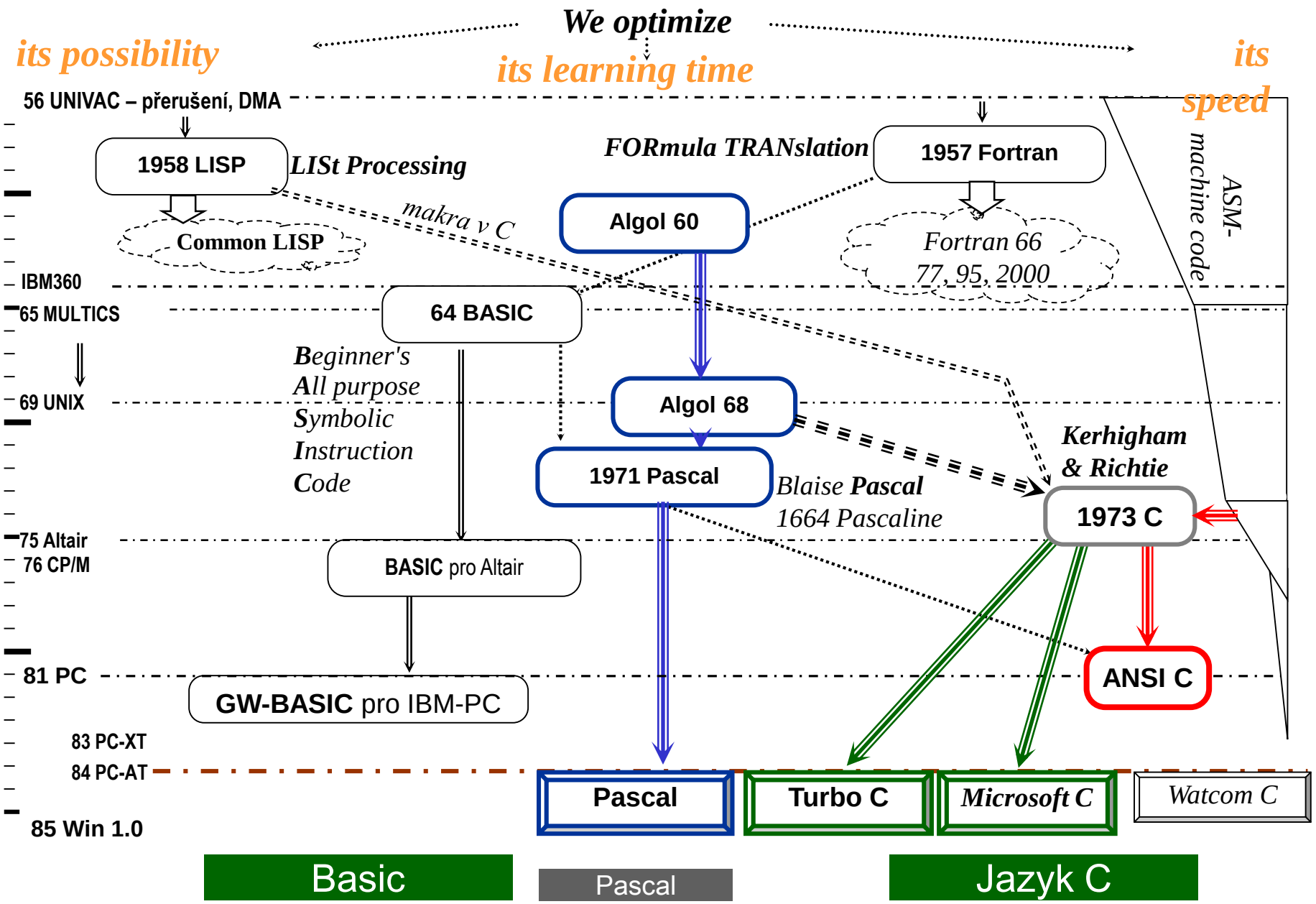
you will later work with this board



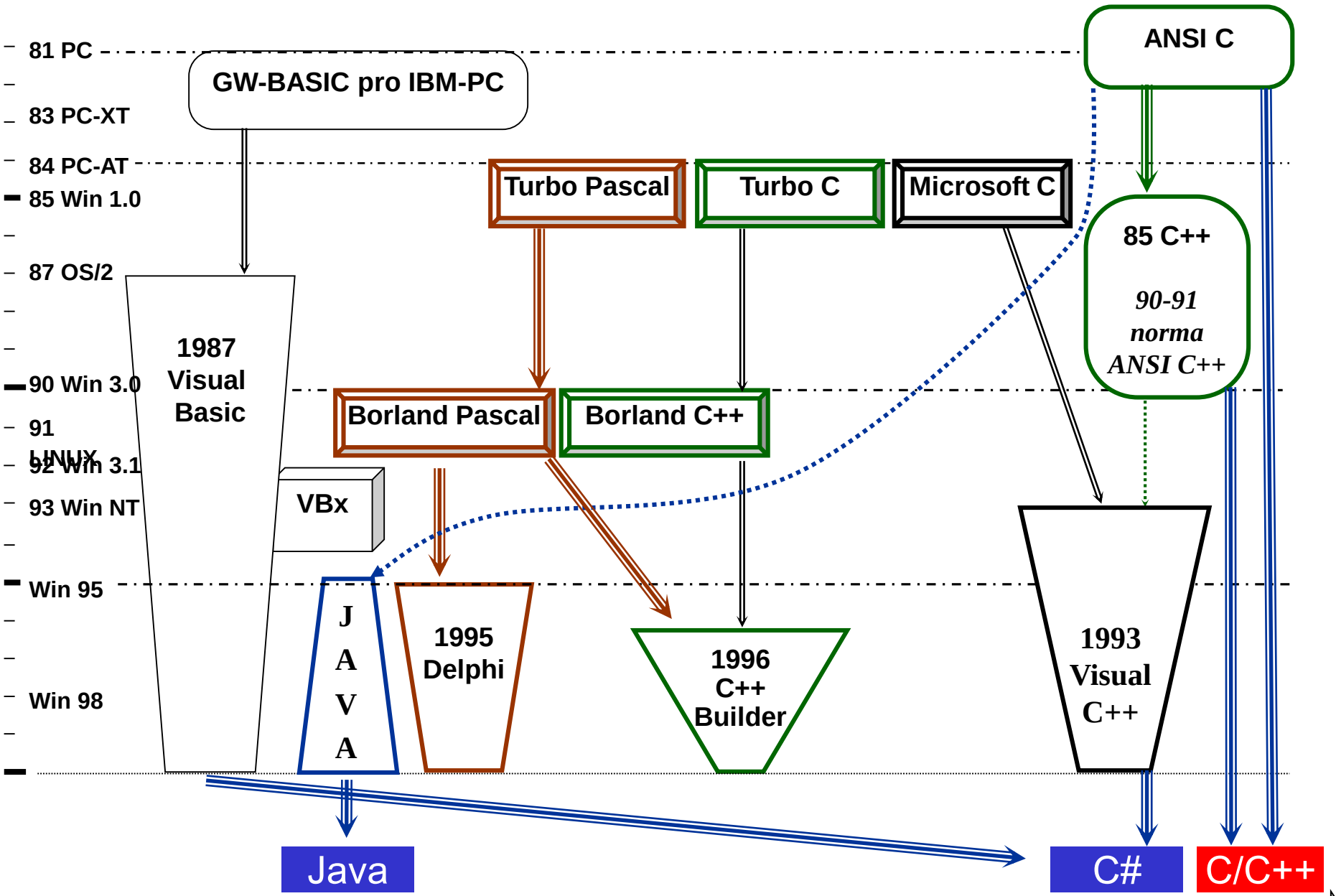
Let's recall ... Physical address space



Searching for Programming Language



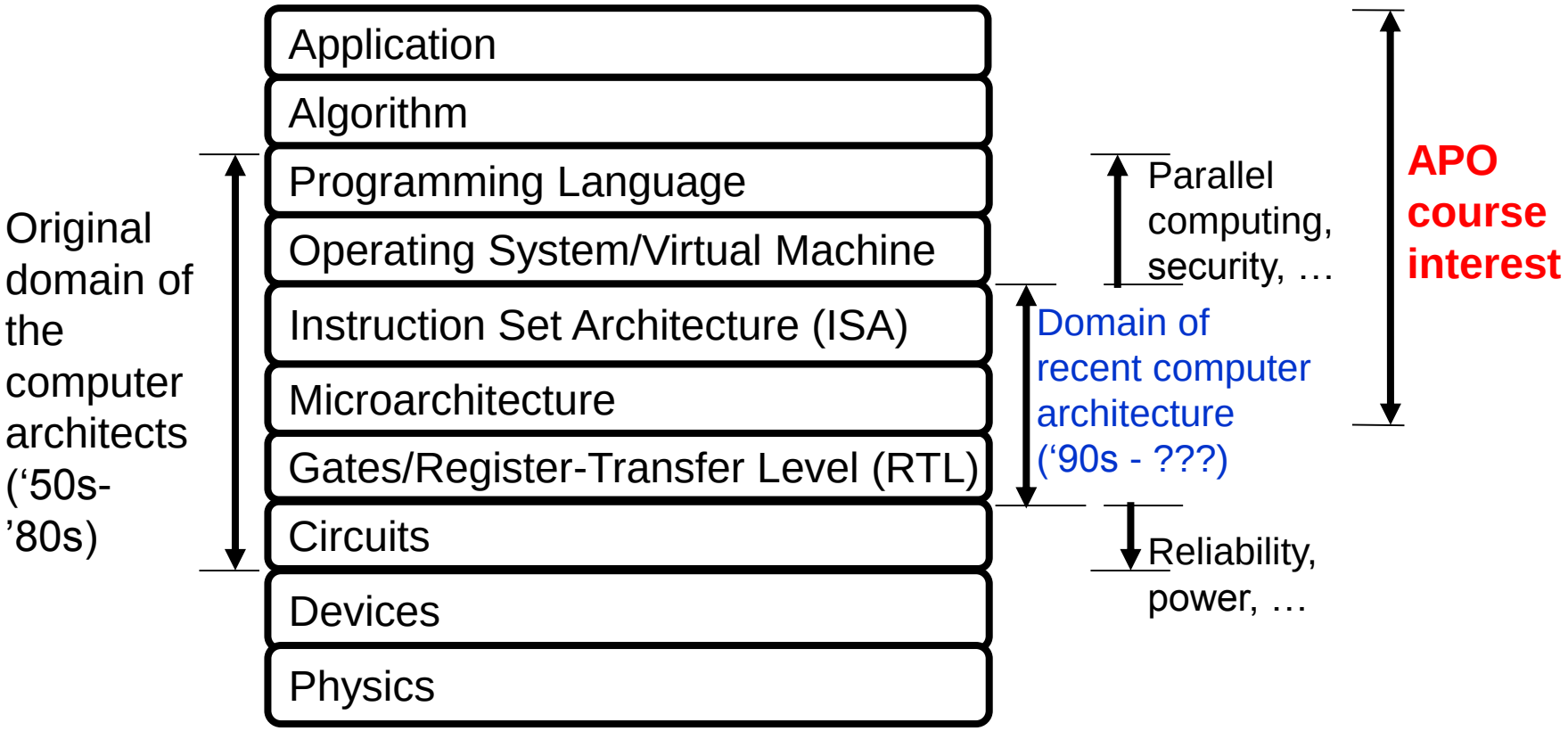
Language C still remains in the game



It is easy to see by formal-logical methods that there exist certain [instruction sets] that are in abstract adequate to control and cause the execution of any sequence of operation. The really decisive considerations from the present point of view, in selecting an [instruction set], are more of a practical nature: simplicity of the equipment demanded by the [instruction set], and ***the clarity of its application to the actually important problems together with the speed of its handling of those problems****.*

[Burks, Goldstine, and von Neumann, 1947]

Computer



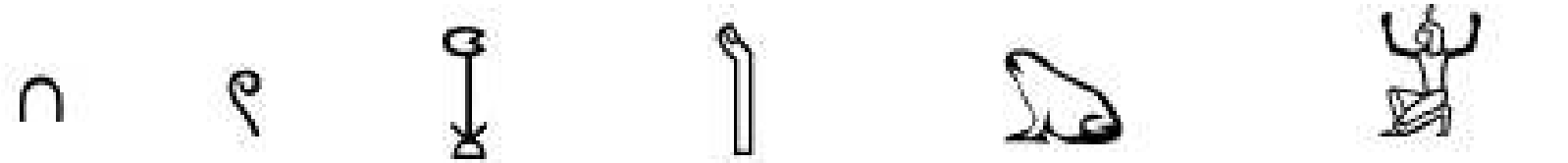
Reference: John Kubiatowicz: EECS 252 Graduate Computer Architecture, Lecture 1. University of California, Berkeley

1st lecture

- How they are stored on your computer
  - INTEGER numbers, with or without sign?
- How to perform basic operations
  - Adding, Subtracting,
  - Multiplying

Non-positional numbers ☺

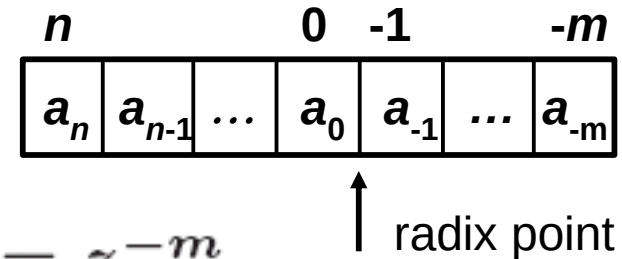
http://diameter.si/sciquest/E1.htm



The value is the sum: 1 333 331

Terminology basics

- Positional (place-value) notation
- Decimal/radix point
- **z** … base of numeral system
- smallest representable number $\varepsilon = z^{-m}$
- **Module** = $\mathcal{Z}$, one increment/unit higher than biggest representable number for given encoding/notation
- **A**, the representable number for given **n** and **m** selection, where **k** is natural number in range ⟦**0,z^{n+m+1} -1**⟧
- The representation and value



$$0 \leq A = k \cdot \varepsilon < \mathcal{Z}$$

$$A \sim a_n a_{n-1} \ldots a_0, a_1 \ldots a_{-m}$$
$$A = a_n z^n + a_{n-1} z^{n-1} + \ldots + a_0 + a_1 z^{-1} \ldots a_{-m} z^{-m}$$

Unsigned integers

Language C:

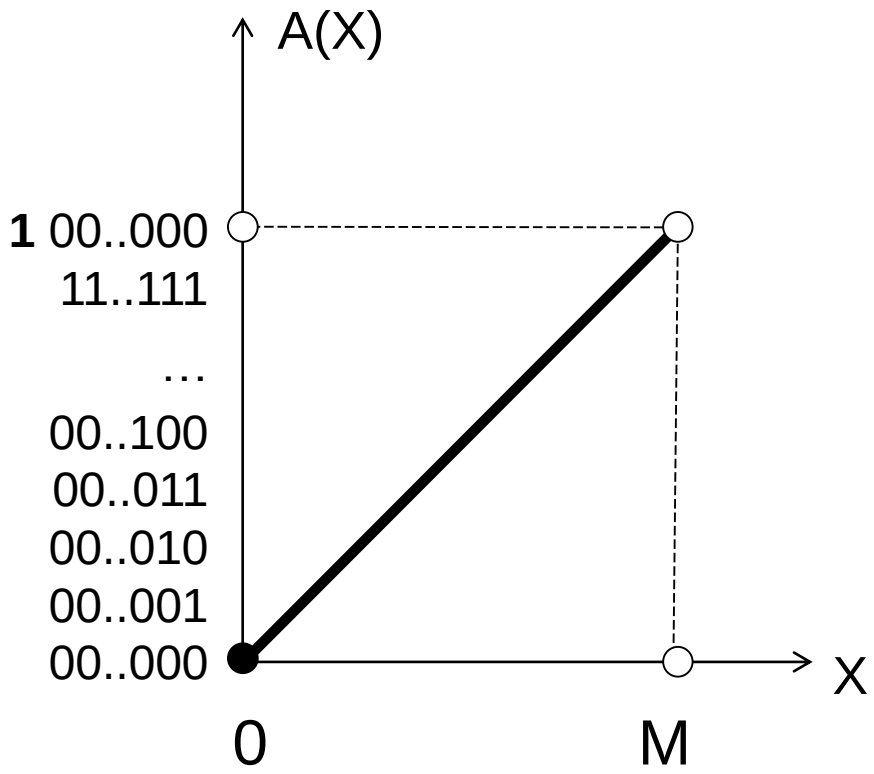
unsigned int

Integer number representation (unsigned, non-negative)

- The most common numeral system base in computers is z=2
- The value of $\mathbf{a_i}$ is in range {0,1,…z-1}, i.e. {0,1} for base 2
- This maps to true/false and unit of information (bit)
- We can represent number **0** … $\mathbf{2^n-1}$ when **n** bits are used
- Which range can be represented by one byte?

  1B (byte) … 8 bits, $2^8 = 256_d$ combinations, values 0 … 255_d = $0b11111111_b$
- Use of multiple consecutive bytes
  - 2B … $2^{16} = 65536_d$, 0 … $65535_d = 0xFFFF_h$,(h … hexadecimal, base 16, a in range 0, … 9, A, B, C, D, E, F)
  - 4B … $2^{32} = 4294967296_d$, 0 … 4294967295_d = $0xFFFFFFFF_h$

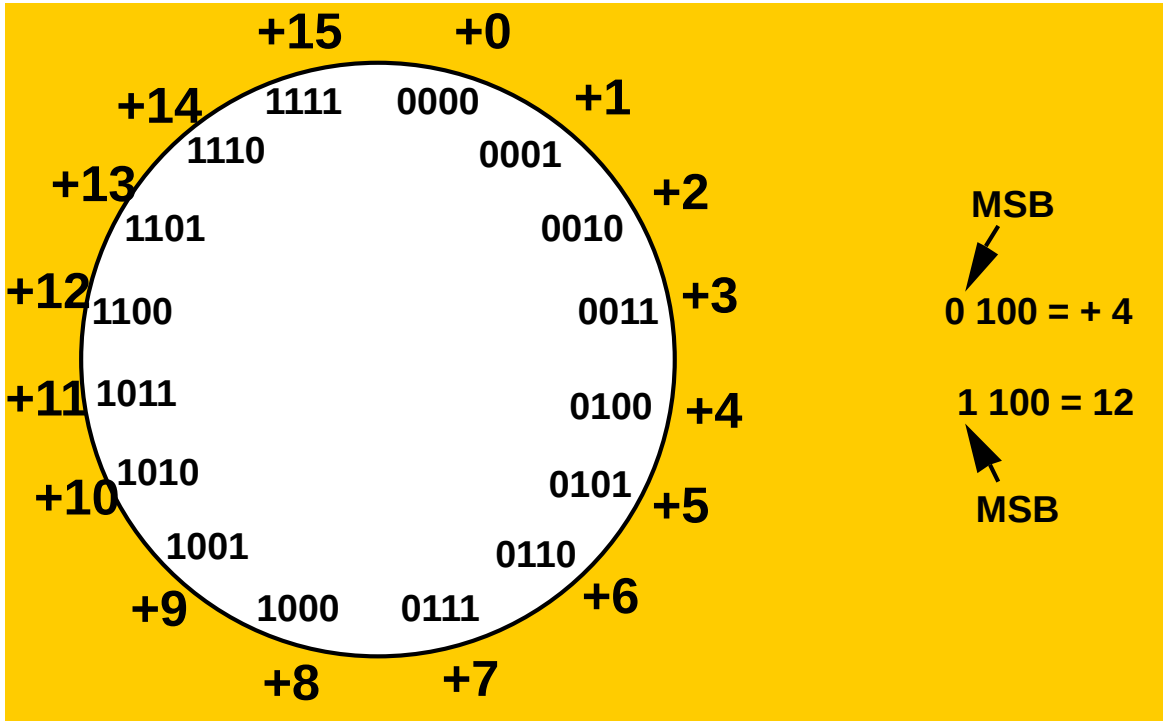
Unsigned integer



binary value	unsigned int
00000000	$0_{(10)}$
00000001	$1_{(10)}$
⋮	⋮
01111101	$125_{(10)}$
01111110	$126_{(10)}$
01111111	$127_{(10)}$
10000000	$128_{(10)}$
10000001	$129_{(10)}$
10000010	$130_{(10)}$
⋮	⋮
11111101	$253_{(10)}$
11111110	$254_{(10)}$
11111111	$255_{(10)}$

Unsigned 4-bit numbers

Assumptions:we'll assume a 4 bit machine word



- Cumbersome subtraction

Signed numbers

Language C:

int

signed int

Two's Complement.

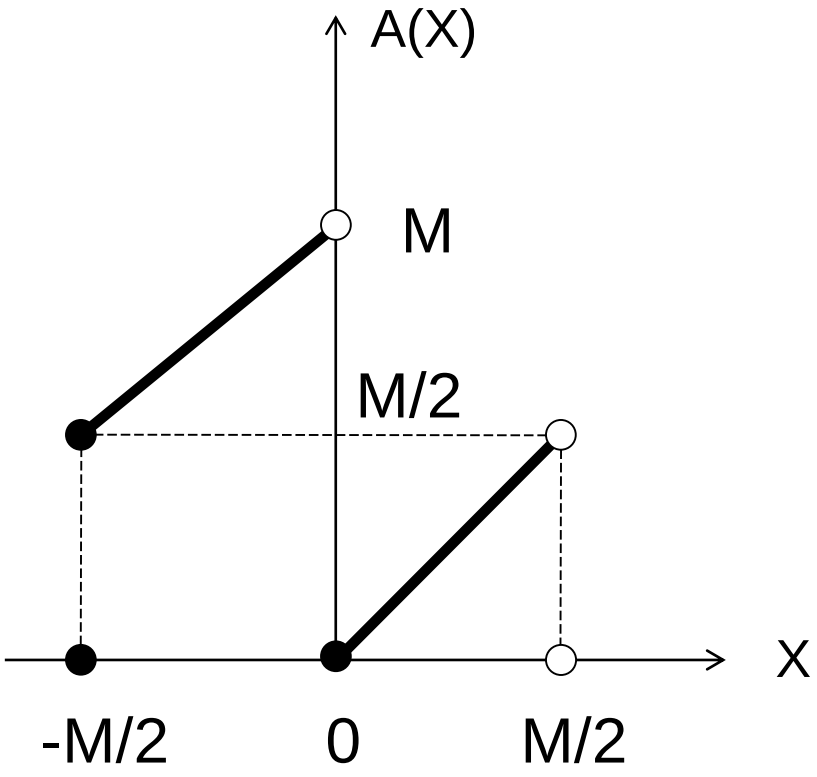
- The most frequent code
- The sum of two opposite numbers with the same absolute value is 00000000H!

Decimal value	4 bit two's compliment
6	0110
-6	1010

Two's Complement

Dvojkový doplněk – pokračování…

- Pokud **N** bude **počet bitů:**
  $<-2^{N-1}, 2^{N-1}-1>$

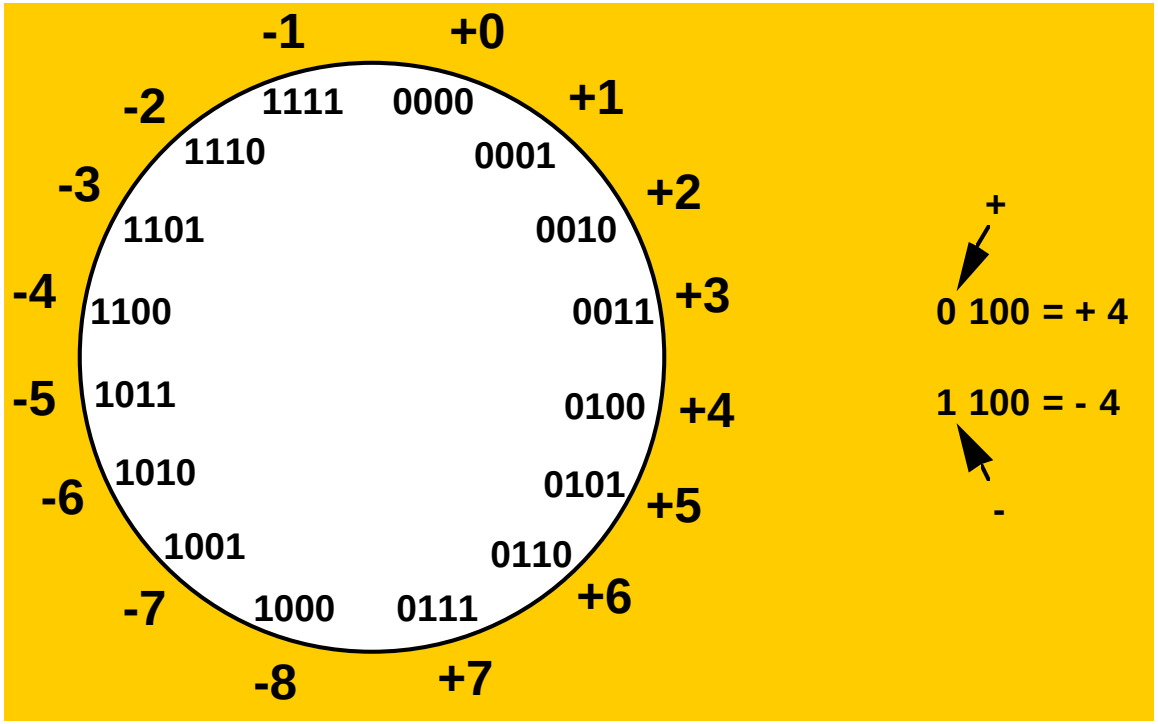


Binární hodnota	Dvojkový doplněk
00000000	$0_{(10)}$
00000001	$1_{(10)}$
⋮	⋮
01111101	$125_{(10)}$
01111110	$126_{(10)}$
01111111	$127_{(10)}$
10000000	$-128_{(10)}$
10000001	$-127_{(10)}$
10000010	$-126_{(10)}$
⋮	⋮
11111101	$-3_{(10)}$
11111110	$-2_{(10)}$
11111111	$-1_{(10)}$

Two's complement - examples

- Examples:
  - $0_D = 00000000_H$,
  - $1_D = 00000001_H$,
  - $2_D = 00000002_H$,
  - $3_D = 00000003_H$,
  - $-1_D = FFFFFFFF_H$,
  - $-2_D = FFFFFFFE_H$,
  - $-3_D = FFFFFFFD_H$,

Twos Complement
(In Czech: Druhý doplněk)



Only one representation for 0

One more negative number than positive number

Two's complement – addition and subtraction

- **Addition**
  - `0000000 0000 0111`$_B \approx 7_D$ Symbols use: `0`=0_H, `0`=0_B
  - `+ 0000000 0000 0110`$_B \approx 6_D$
  - `0000000 0000 1101`$_B \approx 13_D$
- **Subtraction** can be realized as addition of negated number
  - `0000000 0000 0111`$_B \approx 7_D$
  - `+ FFFFFFF 1111 1010`$_B \approx -6_D$
  - `0000000 0000 0001`$_B \approx 1_D$
- Question for revision: how to obtain negated number in two's complement binary arithmetics?

Other Possibilities

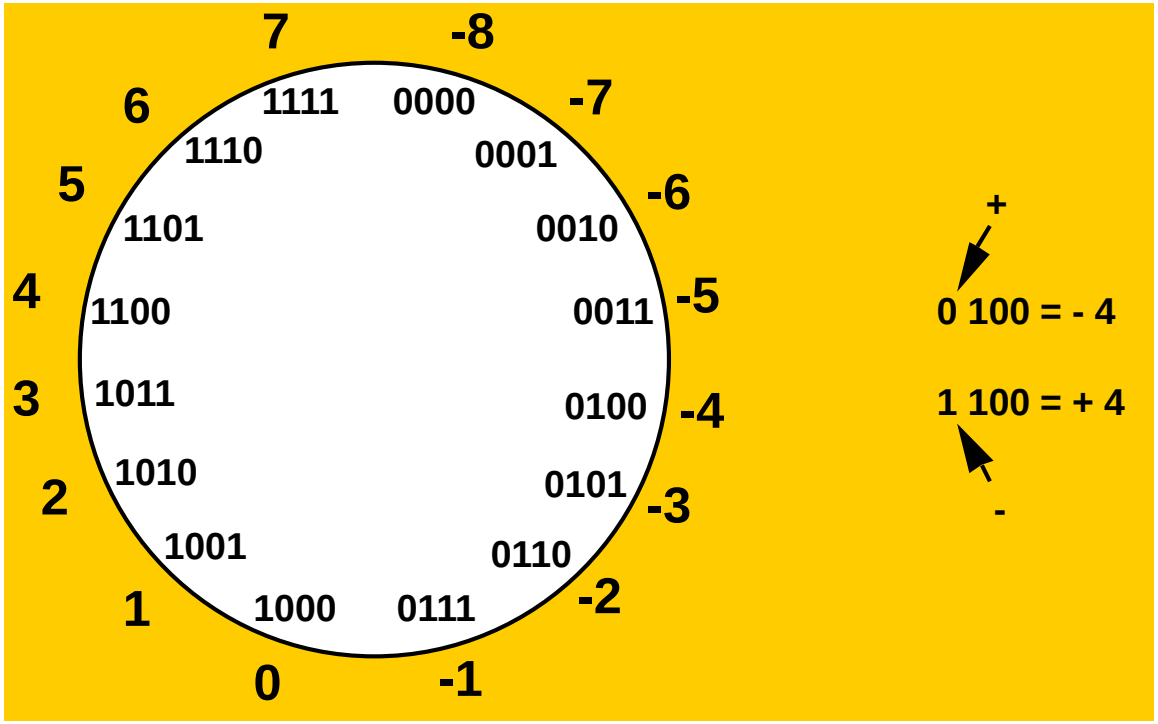
Integer – biased representation

- Known as excess-K or offset binary as well
- Transform to the representation

  $-K \ldots 0 \ldots 2^n-1-K$

  $D(A) = A+K$
- Usually $K=Z/2$
- Advantages
  - Preserves order of original set in mapped set/representation
- Disadvantages
  - Needs adjustment by -K after addition and +K after subtraction processed by unsigned arithmetic unit
  - Requires full transformation before and after multiplication

Excess-K, offset binary or biased representation

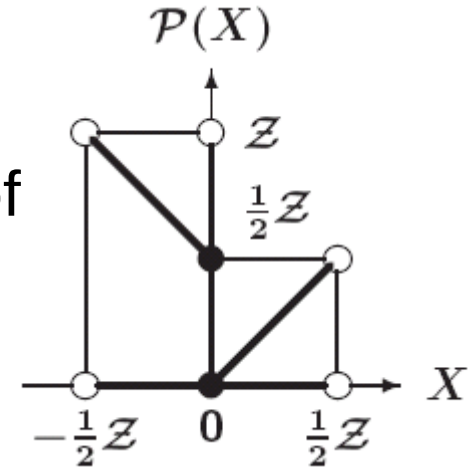


One 0 representation, we can select count of negative numbers - *used e.g. for exponents of real numbers*..

Integer arithmetic unit are not designed to calculate with Excess-K numbers

Integer – sign-magnitude code

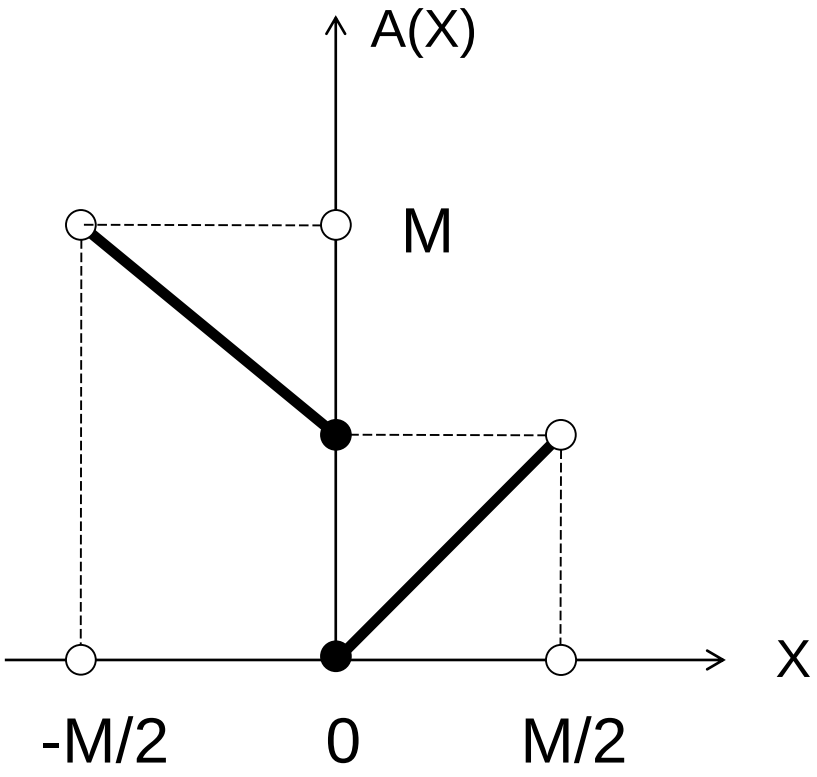
- Sign and magnitude of the value (absolute value)
- Natural to humans -1234, 1234
- One (usually most significant – MSB) bit of the memory location is used to represent the sign
- Bit has to be mapped to meaning
- Common use 0 ≈ “+”, 1 ≈ “-”
- Disadvantages:
  - When location is **k** bits long then only **k-1** bits hold magnitude and each operation has to separate sign and magnitude
  - Two representations of the value 0



$-2^{n-1}+1 \ldots 0 \ldots 2^{n-1}-1$

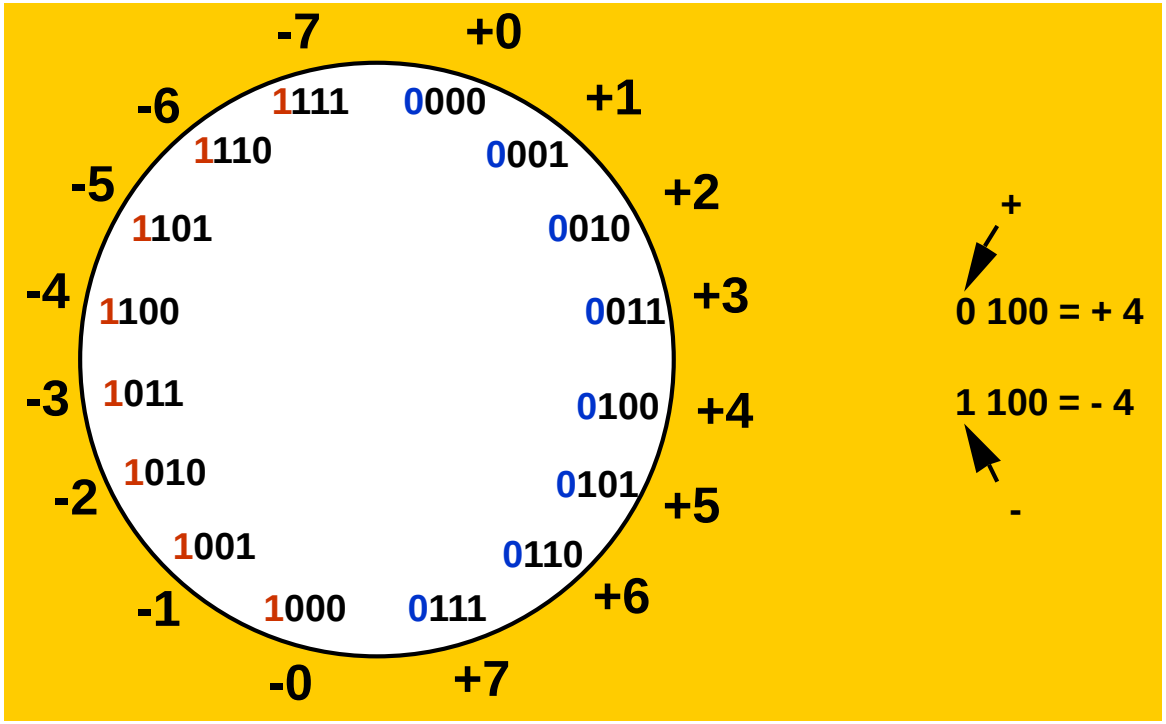
Sign and Magnitude Representation.

$<-2^{N-1}-1,\ 2^{N-1}-1>$



Binary value	Code
00000000	$+0_{(10)}$
00000001	$1_{(10)}$
⋮	⋮
01111101	$125_{(10)}$
01111110	$126_{(10)}$
01111111	$127_{(10)}$
10000000	$-0_{(10)}$
10000001	$-1_{(10)}$
10000010	$-2_{(10)}$
⋮	⋮
11111101	$-125_{(10)}$
11111110	$-126_{(10)}$
11111111	$-127_{(10)}$

Sign and Magnitude Representation



- Cumbersome addition/subtraction
- *Sign+Magnitude usually used only for float point numbers*

Integers – ones' complement

- Transform to the representation

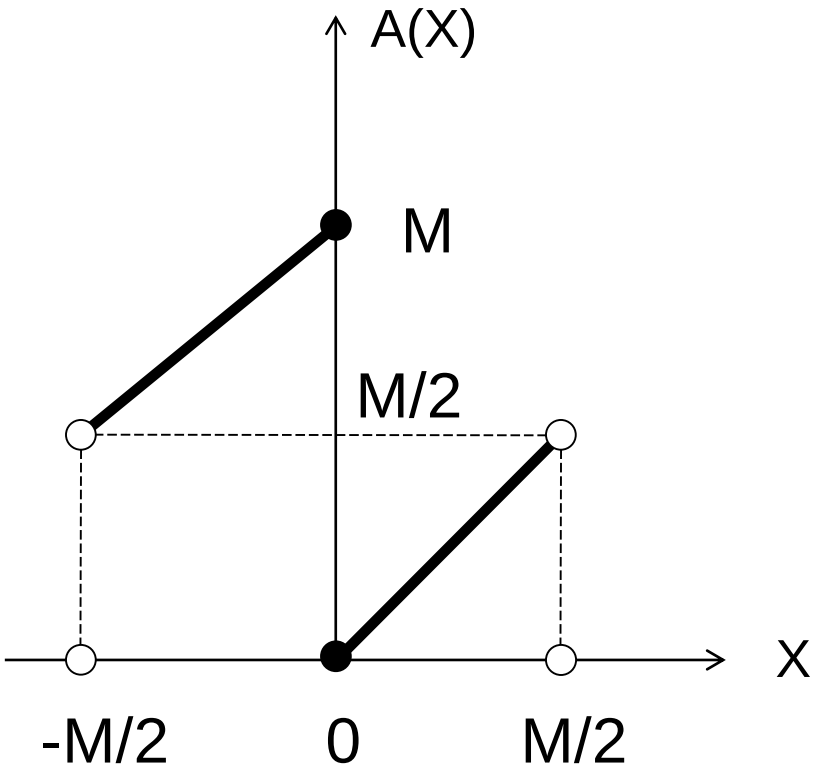
  $-2^{n-1}+1 \ldots 0 \ldots 2^{n-1}-1$

  D(A) = A iff A≥0

  D(A) = Z-1-|A| iff A<0 (i.e. subtract from all ones)
- Advantages
  - Symmetric range
  - Almost continuous, requires hot one addition when sign changes
- Disadvantage
  - Two representations of value 0
  - More complex hardware
- Negate (-A) value can be computed by bitwise complement (flipping) of each bit in representation

Ones Complement

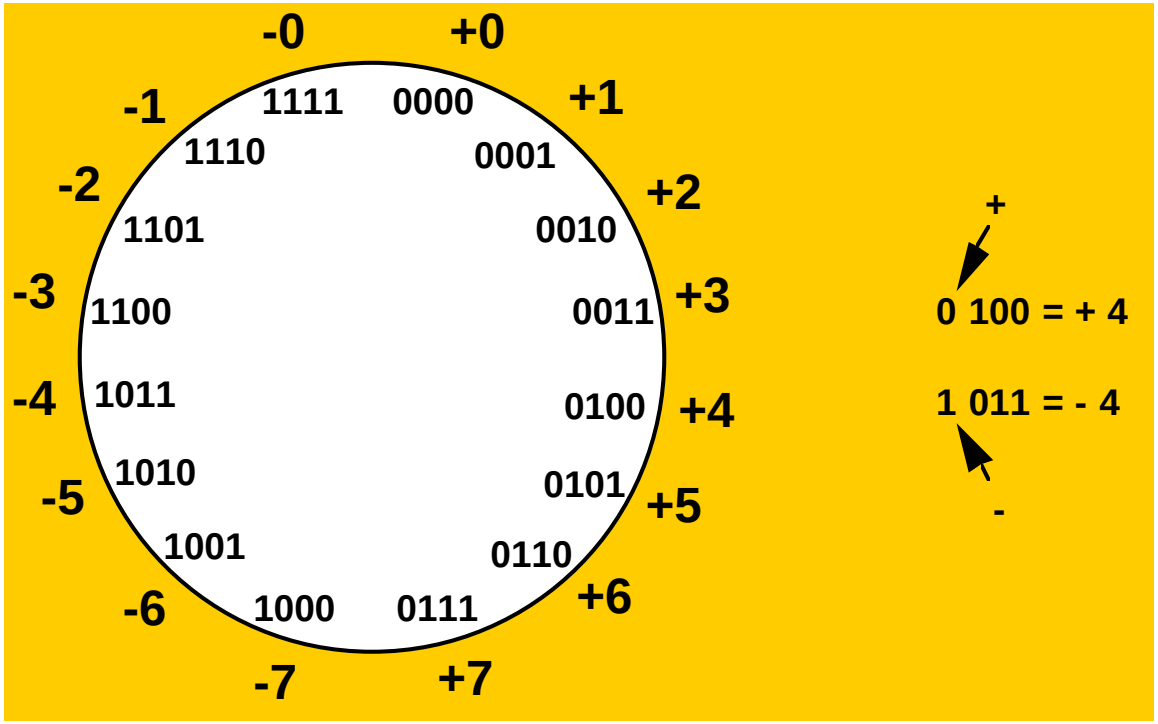
$<-2^{N-1} -1, 2^{N-1} -1>$



Binary value	Code
00000000	$0_{(10)}$
00000001	$1_{(10)}$
⋮	⋮
01111101	$125_{(10)}$
01111110	$126_{(10)}$
01111111	$127_{(10)}$
10000000	$-127_{(10)}$
10000001	$-126_{(10)}$
10000010	$-125_{(10)}$
⋮	⋮
11111101	$-2_{(10)}$
11111110	$-1_{(10)}$
11111111	$-0_{(10)}$

Ones Complement

(In Czech: První doplněk)



Still two representations of 0! This causes some problems

Some complexities in addition, nowadays nearly not used

OPERATION WITH INTEGERS

Direct realization of adder as logical function

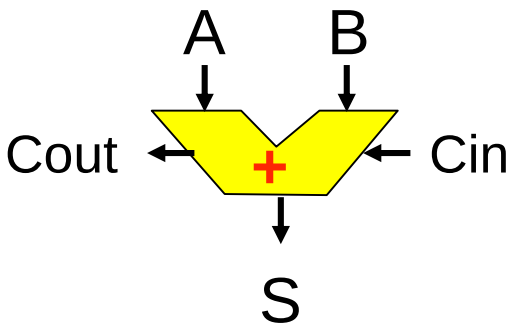
bit width	Number of logic operations for calculating sum
1	3
2	22
3	89
4	272
5	727
6	1567
7	3287
8	7127
9	17623
10	53465
11	115933

Complexity is higher than $O(2^n)$

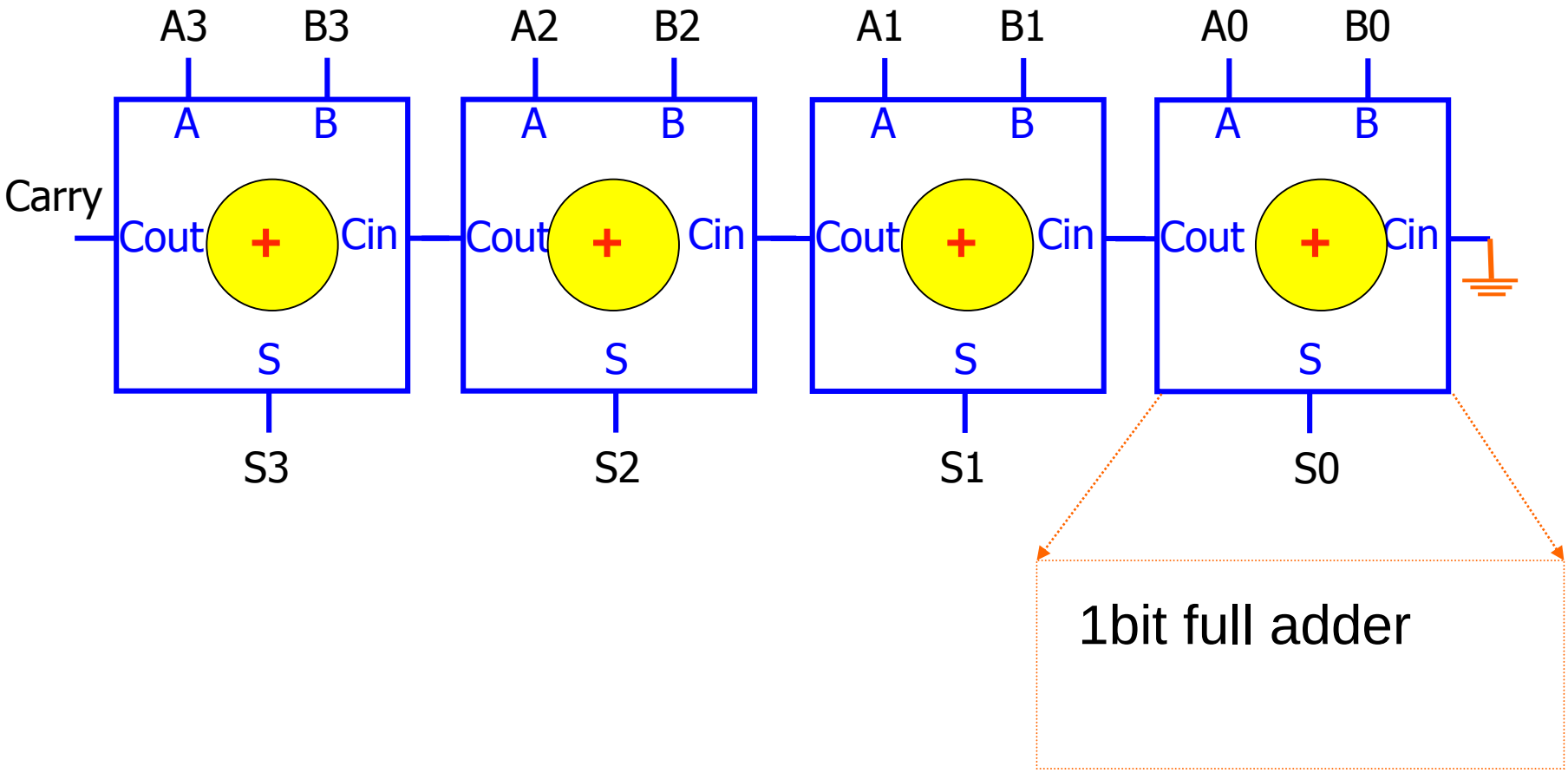
The calculation was performed by BOOM logic minimizer created at the Department of Computer Science CTU-FEE

1bit Full Adder

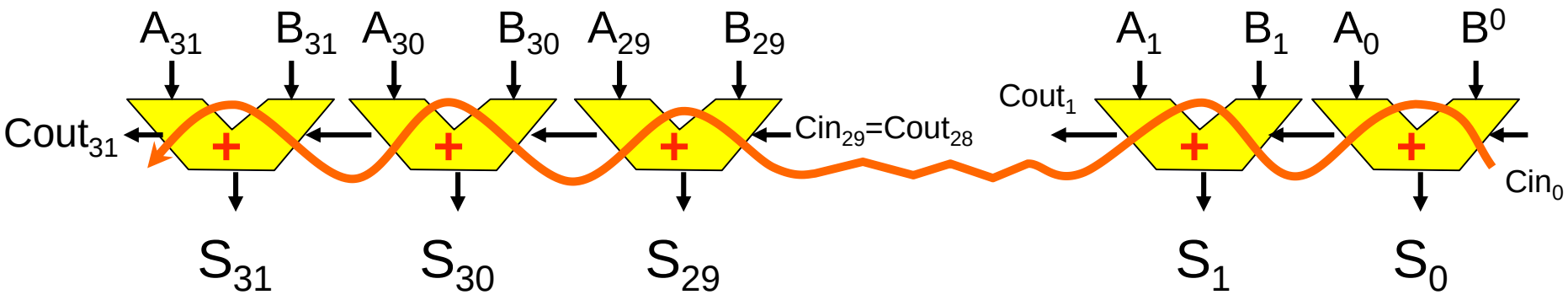
A	0	0	1	1	0	0	1	1
+B	0	1	0	1	0	1	0	1
Sum	00	01	01	10	00	01	01	10
+ Carry-In	0	0	0	0	1	1	1	1
CarryOut Sum	00	01	01	10	01	10	10	11



Adder


A3
B3
A2
B2
A1
B1
A0
B0
A
B
Carry
Cout
+
Cin
S
S3
S2
S1
S0
1bit full adder

Simple Adder

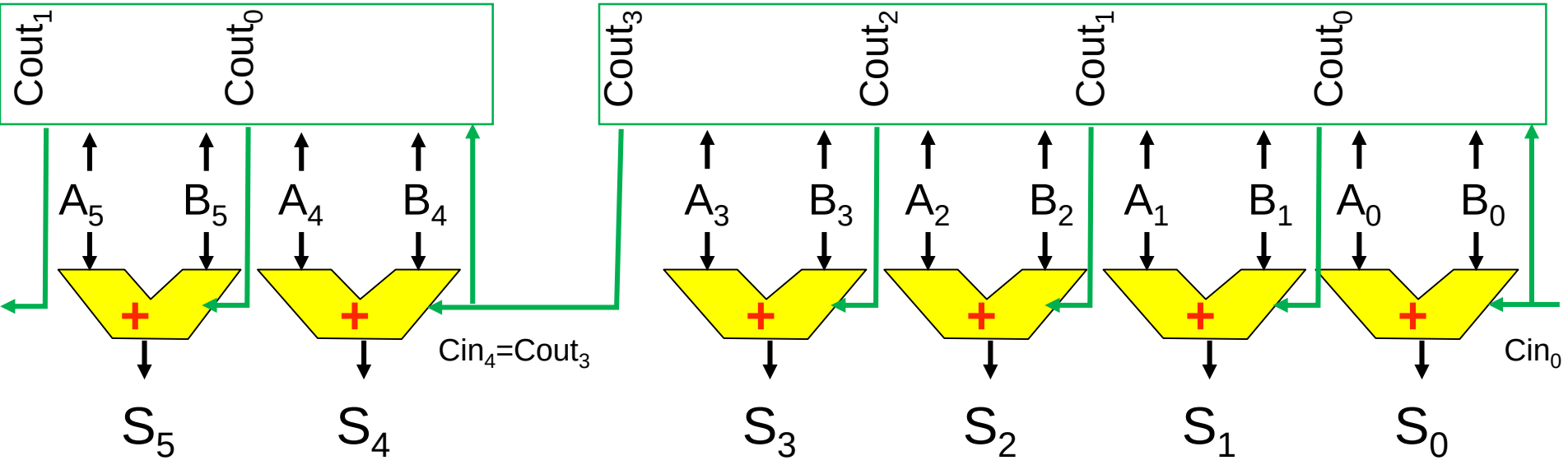


Simplest N-bit adder

- we chain 1-bit full adders
- "Carry" ripple through their chain
- Minimal number of logical elements
- Delay is given by the last Cout - 2*(N-1)+ 3 gates of the last adder
  = (2 N+1) times propagation delay of 1 gate

32bit CLA "carry look-ahead" adder

The carry-lookahead adder calculates one or more carry bits before the sum, which reduces the wait time to calculate the result of the larger value bits



Static "carry look ahead (CLA)" unit for 4 bits

Increment / Decrement

Very fast operations that do not need an adder!

The last bit is always negated, and the previous ones are negated according to the end 1 / 0

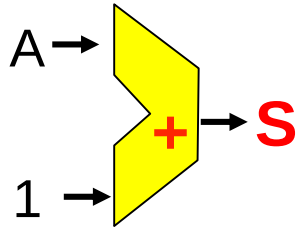
Dec.	Binary 8 4 2 1	+1	Binary 8 4 2 1	-1
0	0000	0001	0000	1111
1	0001	0010	0001	0000
2	0010	0011	0010	0001
3	0011	0100	0011	0010
4	0100	0101	0100	0011
5	0101	0110	0101	0100
6	0110	0111	0110	0101
7	0111	1000	0111	0110
8	1000	1001	1000	0111
9	1001	1010	1001	1000
10	1010	1011	1010	1001
11	1011	1100	1011	1010
12	1100	1101	1100	1011
13	1101	1110	1101	1100
14	1110	1111	1110	1101
15	1111	0000	1111	1110

Special Case +1/-1

S0=not A0

S1=A1 xor A0

S2=A2 xor (A1 and A0)

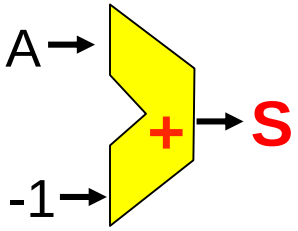


Eq: $S_i = A_i$ xor (A_{i-1} and A_{i-2} and … A_1 and A_0); i=0..n-1

S0=not A0

S1=A1 xor (not A0)

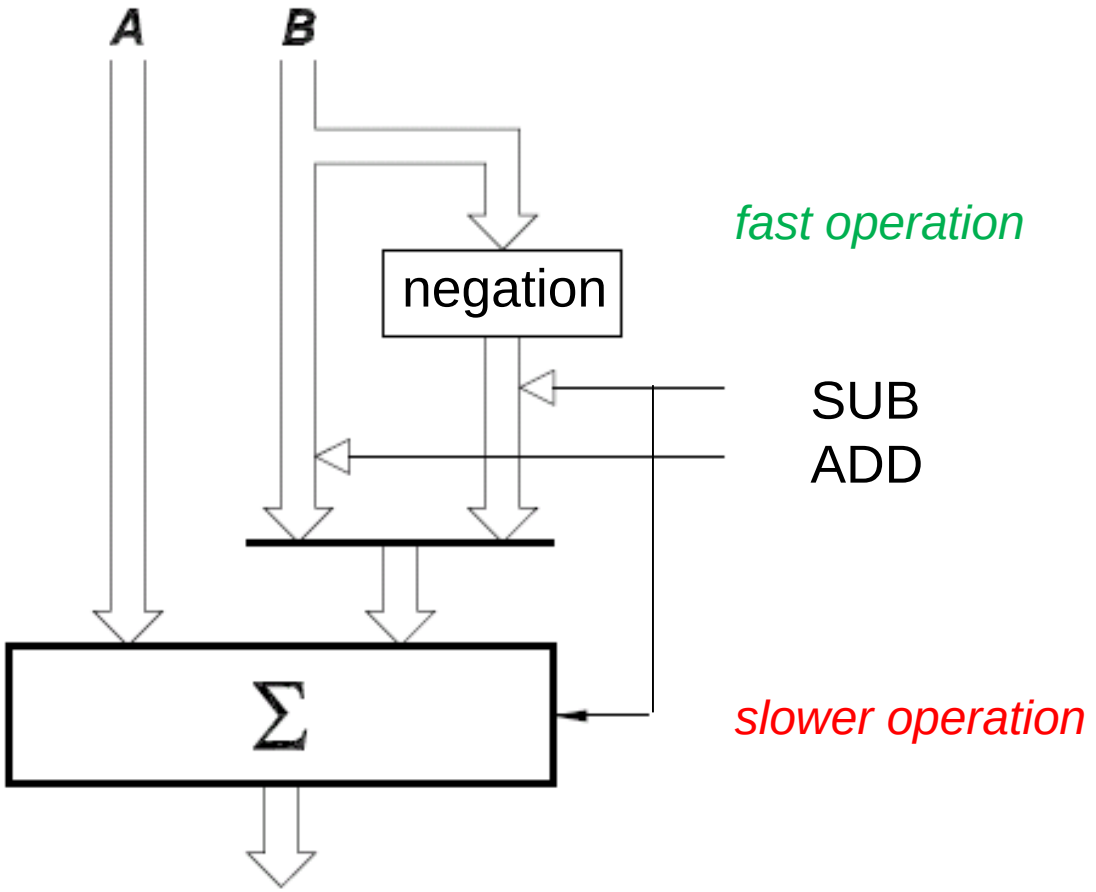
S2=A2 xor (not A1 and not A0)



Eq: $S_i = A_i$ xor (not A_{i-1} and … and not A_0); i=0..n-1

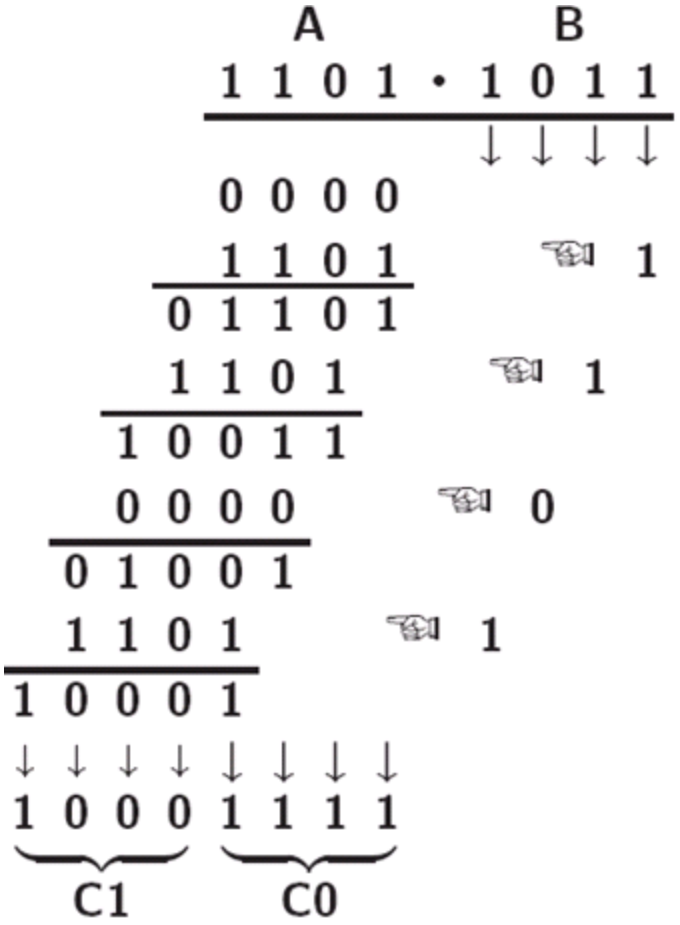
The number of circuits is given by the arithmetic series, with the complexity O (n^2) where n is the number of bits. The operation can be performed in parallel for all bits, and for the both +1/-1 operations, we use a circuit that differs only by negations.

Addition / Subtraction HW

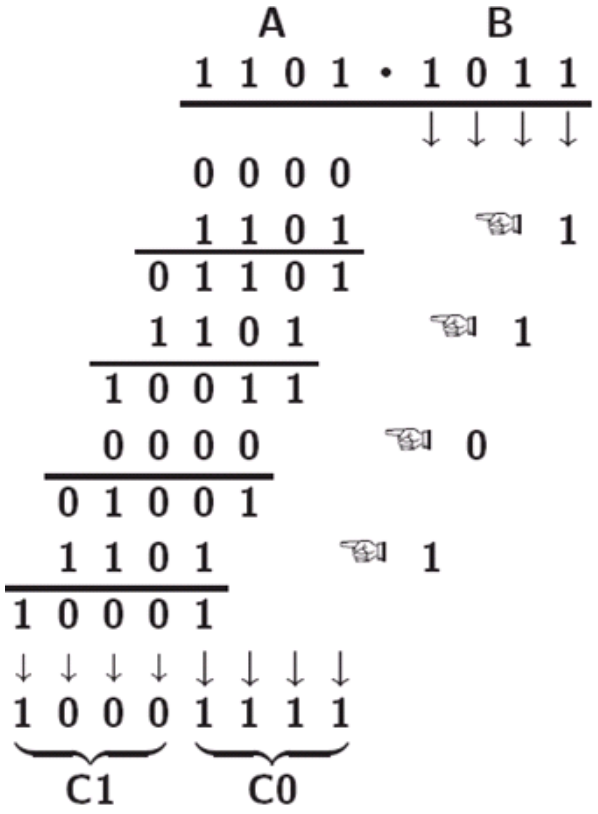


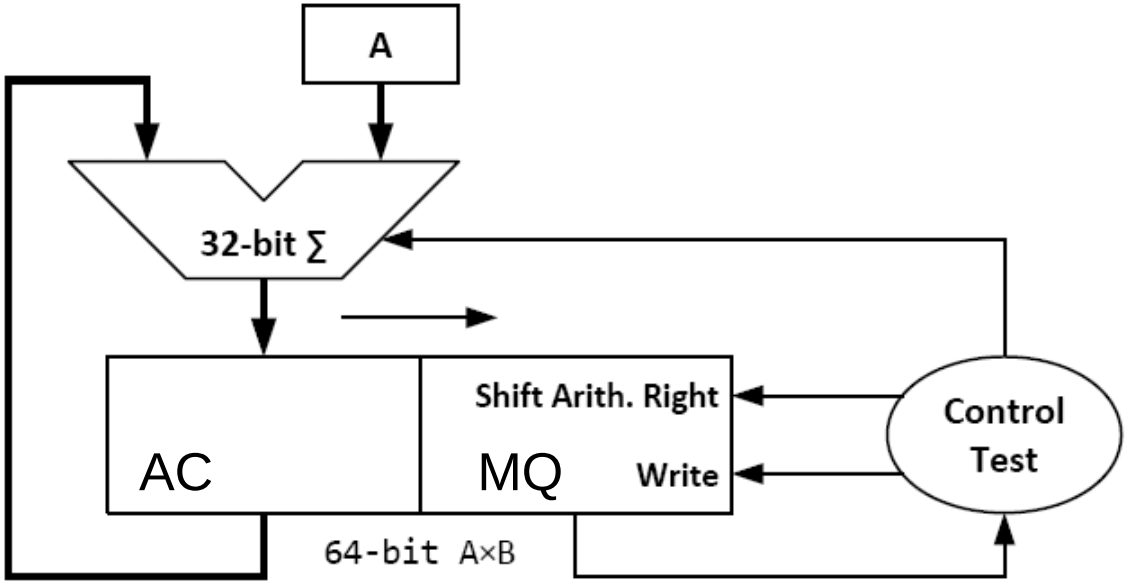
Source: X36JPO, A. Pluháček

Unsigned binary numbers multiplication



Sequential hardware multiplier (32b case)





The speed of the multiplier is horrible

Algorithm for Multiplication

A = multiplicand;
MQ = multiplier;
AC = 0;

for(int i=1; i <= *n*; i++) // *n* – represents number of bits
{
  if(MQ_0 = = 1) AC = AC + A; // MQ_0 = LSB of MQ

  SR (shift AC MQ by one bit right and insert information about carry from the MSB from previous step)
}
end.

when loop ends AC MQ holds 64-bit result

Example of the multiply X by Y

Multiplicand x=110 and multiplier y=101.

i	operation	AC	MQ	A	comment
		000	101	110	initial setup
1	AC = AC+MB	110	101		start of the cycle
	SR	011	010		
2	nothing	011	010		because of MQ_0 = = 0
	SR	001	101		
3	AC = AC+MB	111	101		
	SR	011	110		end of the cycle

The whole operation: x×y = 110×101 = 011110, (6×5 = 30)

Multiplication in two's compliment

Can be implemented, but there is a problem ...

The intended product is generally not the same as the product of two's numbers!

Details are already outside the intended APO range.

The best way is the multiplication of their absolute values and decision about its sign.

Wallace tree based multiplier

Q=X .Y, X and Y are considered as and 8bit unsigned numbers

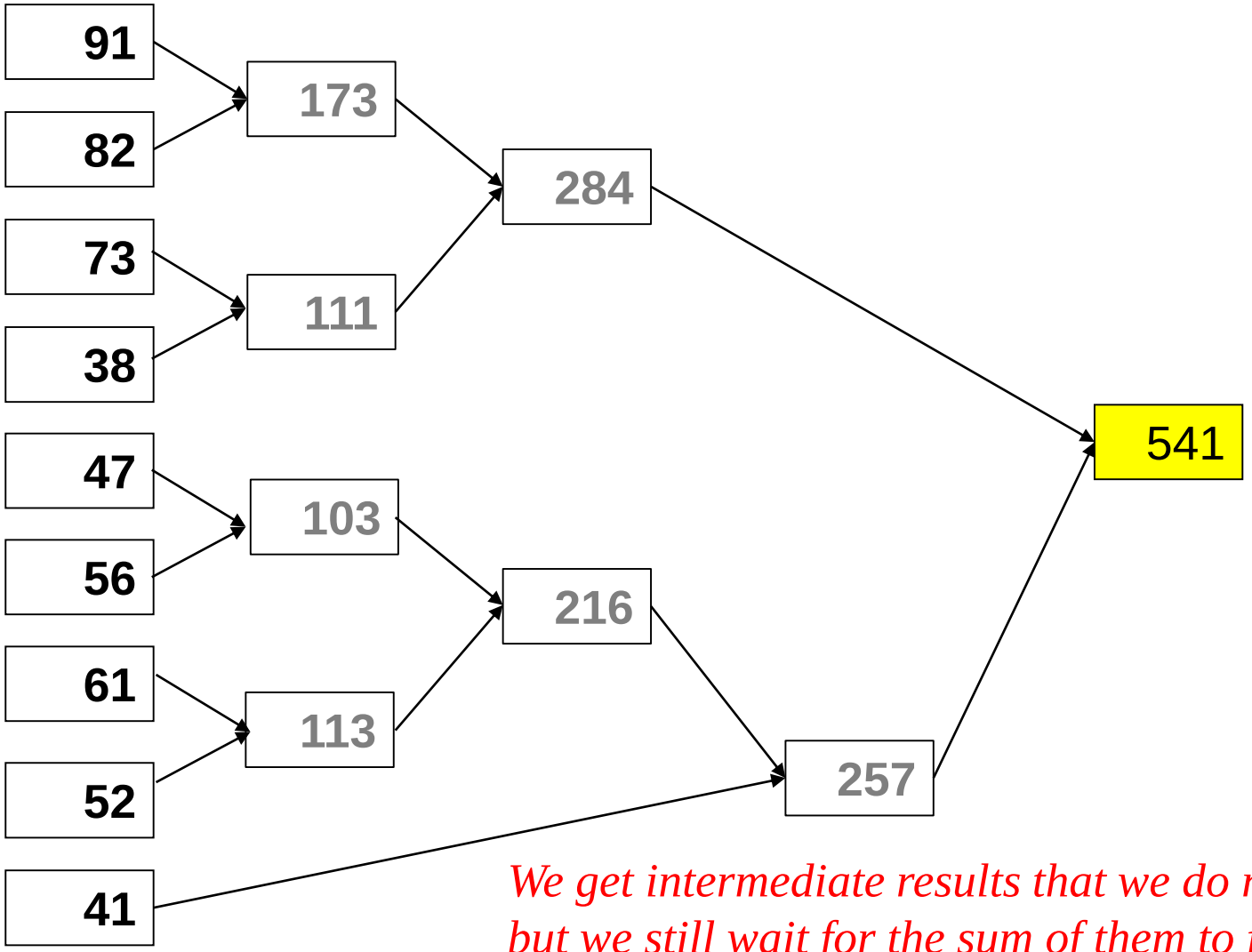
(x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0). (y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0) =

0	0	0	0	0	0	0	0	x_7y_0	x_6y_0	x_5y_0	x_4y_0	x_3y_0	x_2y_0	x_1y_0	x_0y_0	P0
0	0	0	0	0	0	0	x_7y_1	x_6y_1	x_5y_1	x_4y_1	x_3y_1	x_2y_1	x_1y_1	x_0y_1	0	P1
0	0	0	0	0	0	x_7y_2	x_6y_2	x_5y_2	x_4y_2	x_3y_2	x_2y_2	x_1y_2	x_0y_2	0	0	P2
0	0	0	0	0	x_7y_3	x_6y_3	x_5y_3	x_4y_3	x_3y_3	x_2y_3	x_1y_3	x_0y_3	0	0	0	P3
0	0	0	0	x_7y_4	x_6y_4	x_5y_4	x_4y_4	x_3y_4	x_2y_4	x_1y_4	x_0y_4	0	0	0	0	P4
0	0	0	x_7y_5	x_6y_5	x_5y_5	x_4y_5	x_3y_5	x_2y_5	x_1y_5	x_0y_5	0	0	0	0	0	P5
0	0	x_7y_6	x_6y_6	x_5y_6	x_4y_6	x_3y_6	x_2y_6	x_1y_6	x_0y_6	0	0	0	0	0	0	P6
0	x_7y_7	x_6y_7	x_5y_7	x_4y_7	x_3y_7	x_2y_7	x_1y_7	x_0y_7	0	0	0	0	0	0	0	P7
Q_{15}	Q_{14}	Q_{13}	Q_{12}	Q_{11}	Q_{10}	Q_9	Q_8	Q_7	Q_6	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0	

The sum of P0+P1+...+P7 gives result of X and Y multiplication.

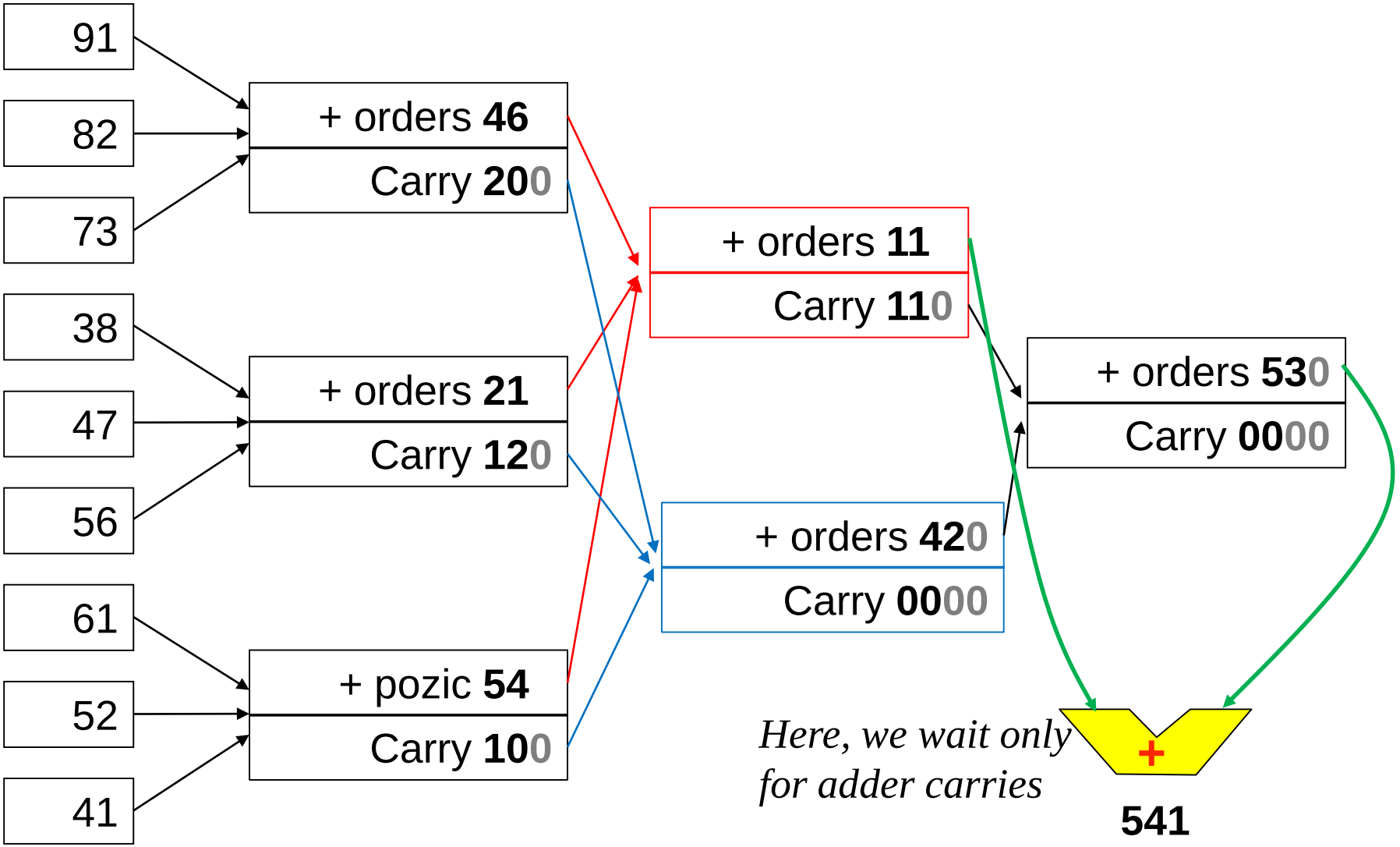
Q = X .Y = P0 + P1 + ... + P7

Parallel adder of 9 numbers



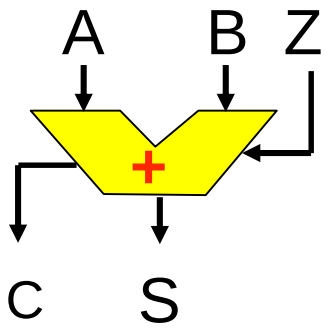
We get intermediate results that we do not need at all, but we still wait for the sum of them to finish!

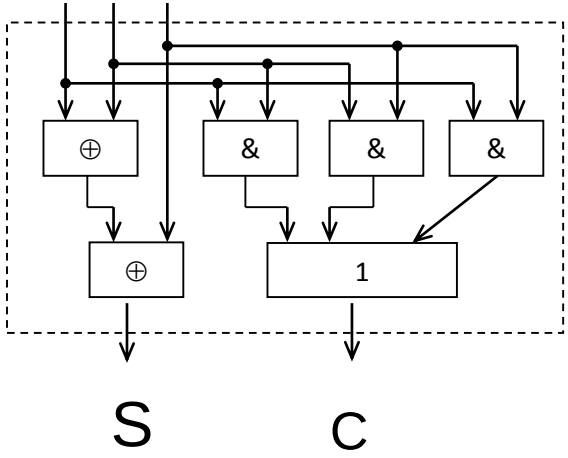
Decadic Carry-save adder



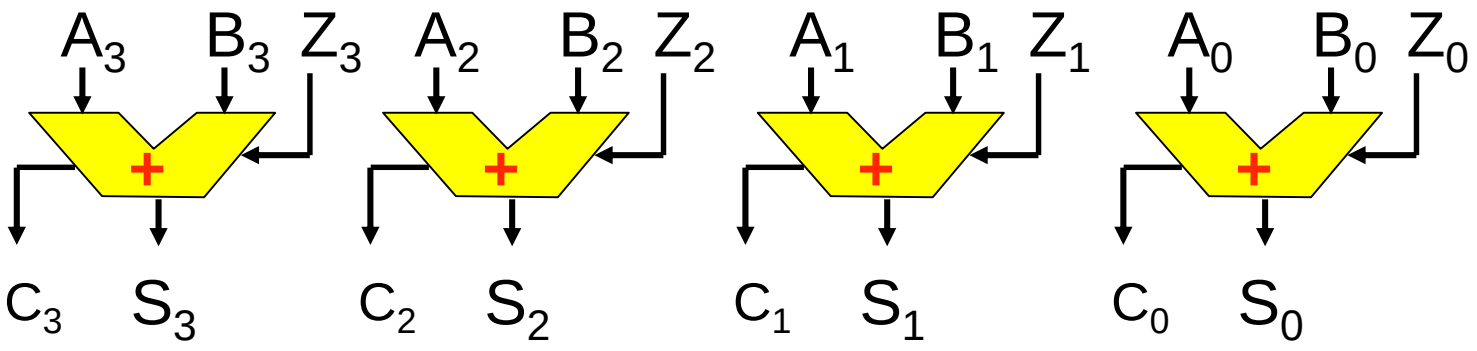
1bit Carry Save Adder

A	0	0	1	1	0	0	1	1
+B	0	1	0	1	0	1	0	1
Z=Carry-In	0	0	0	0	1	1	1	1
Sum	0	1	1	0	1	0	0	1
C=Cout	0	0	0	1	0	1	1	1



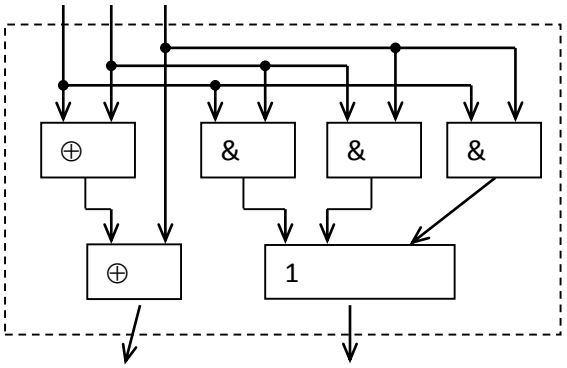


3-bit Carry-save adder



Wallace tree based fast multiplier

The basic element is an CSA circuit (Carry Save Adder)



$$S = S^b + C$$

$$S^b_i = x_i \oplus y_i \oplus z_i$$

$$C_{i+1} = x_i y_i + y_i z_i + z_i x_i$$