Computer Architectures

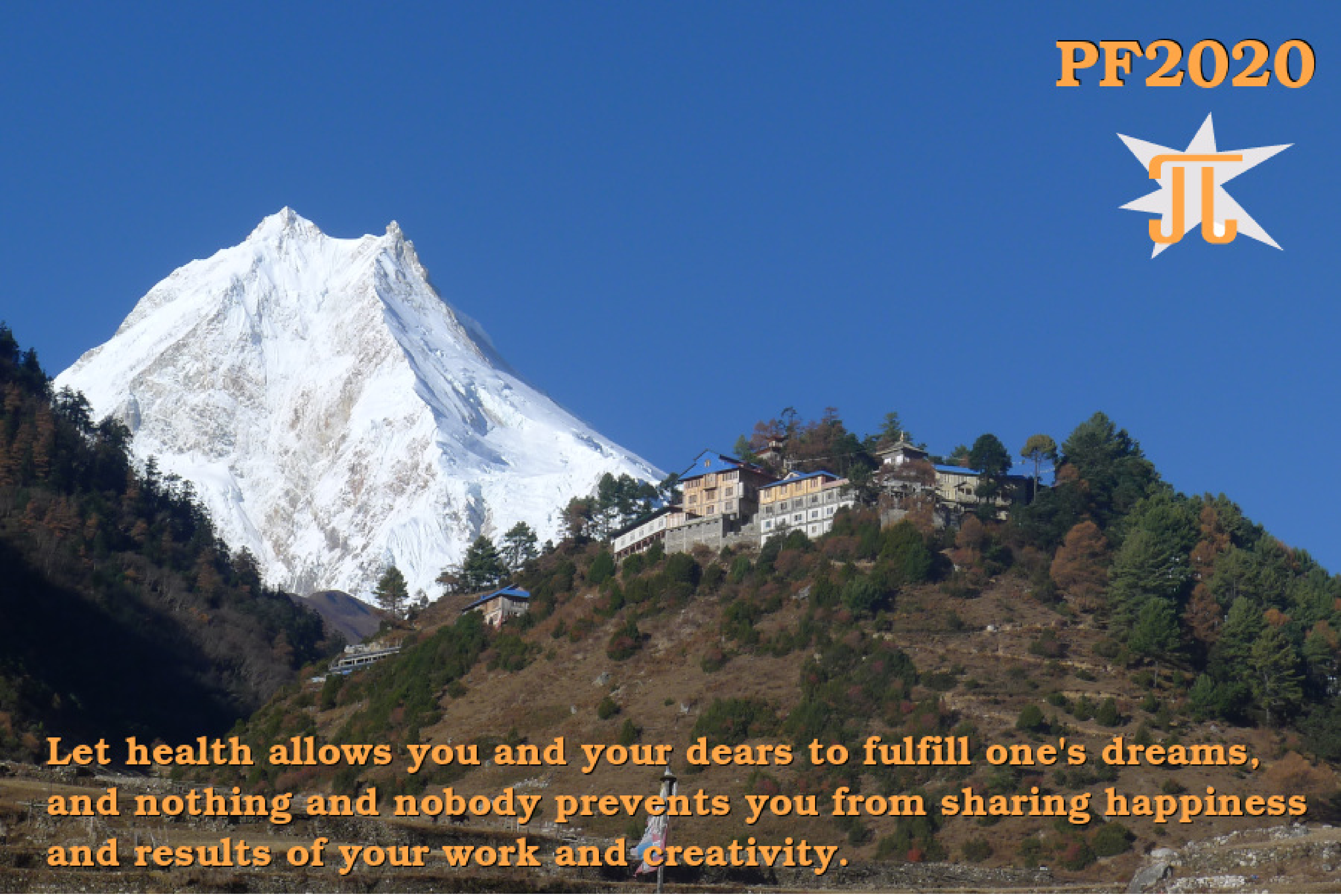
Number Representation and Computer Arithmetics

Pavel Píša, Richard Šusta

Michal Štepanovský, Miroslav Šnorek

Czech Technical University in Prague, Faculty of Electrical Engineering

English version partially supported by:
European Social Fund Prague & EU: We invests in your future.


PF2020
Let health allows you and your dears to fulfill one's dreams,
and nothing and nobody prevents you from sharing happiness
and results of your work and creativity.

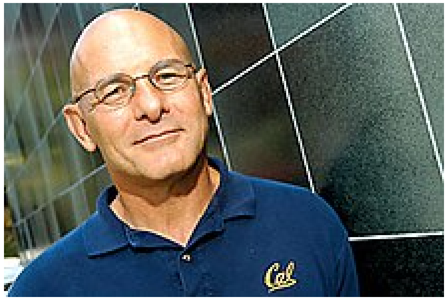
Important Introductory Note

- 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/b192/courses/b35apo/
  https://dcenet.felk.cvut.cz/apo/ *- they will be opened*
- Some followup related subjects:
  - 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.

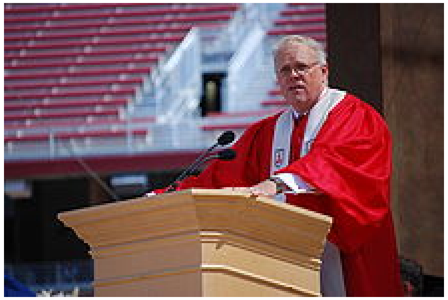
Important Introductory Note

- 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 Berkley RISC → SPARC, DLX, RAID, Clusters, RISC-V

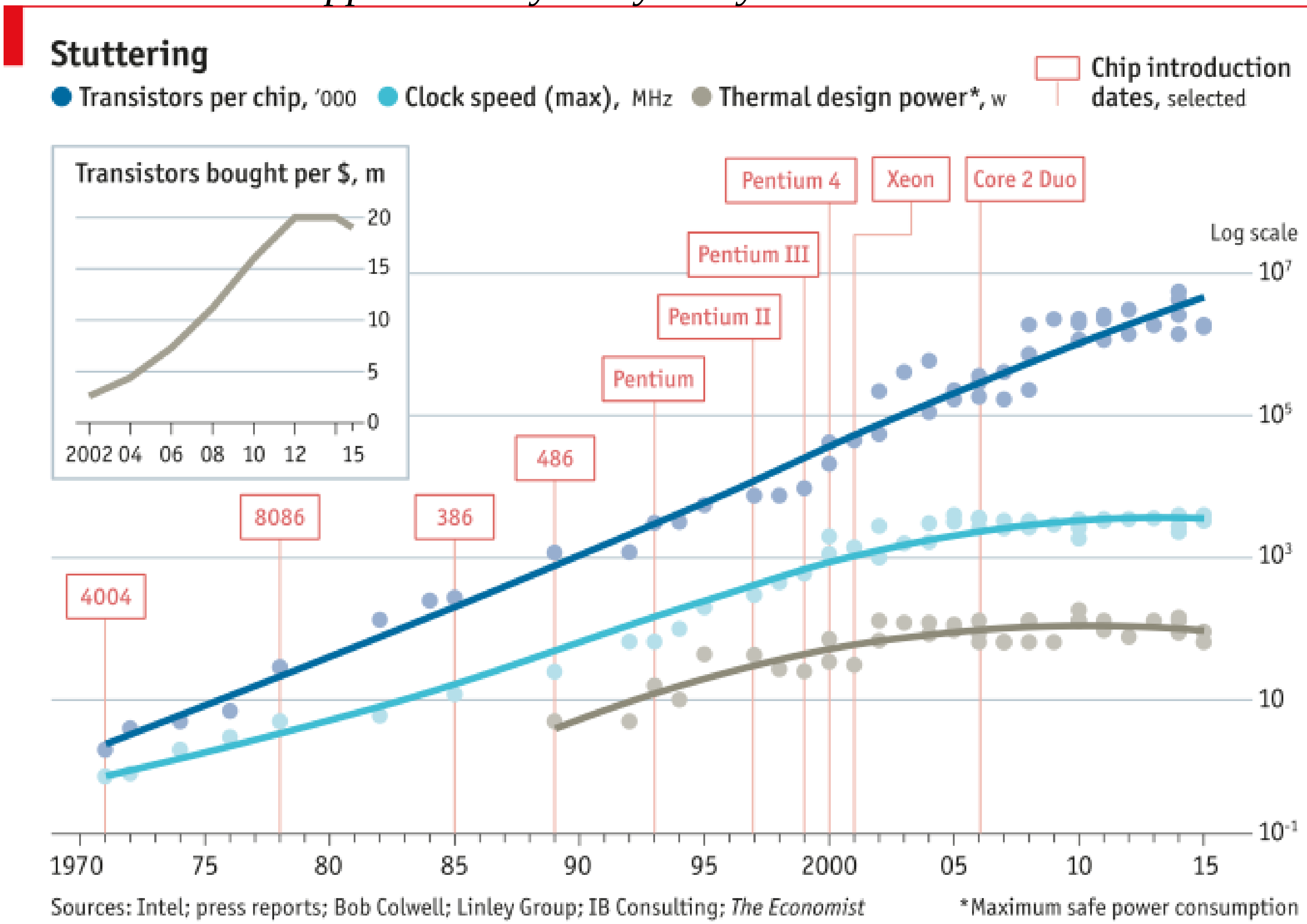


John Leroy Hennessy
10th President of Stanford University
Works: RISC processors MIPS, DLX a MMIX

2017 Turing Award for pioneering a systematic, quantitative approach to the design and evaluation of computer architectures with enduring impact on the microprocessor industry. → A New Golden Age for Computer Architecture – RISC-V

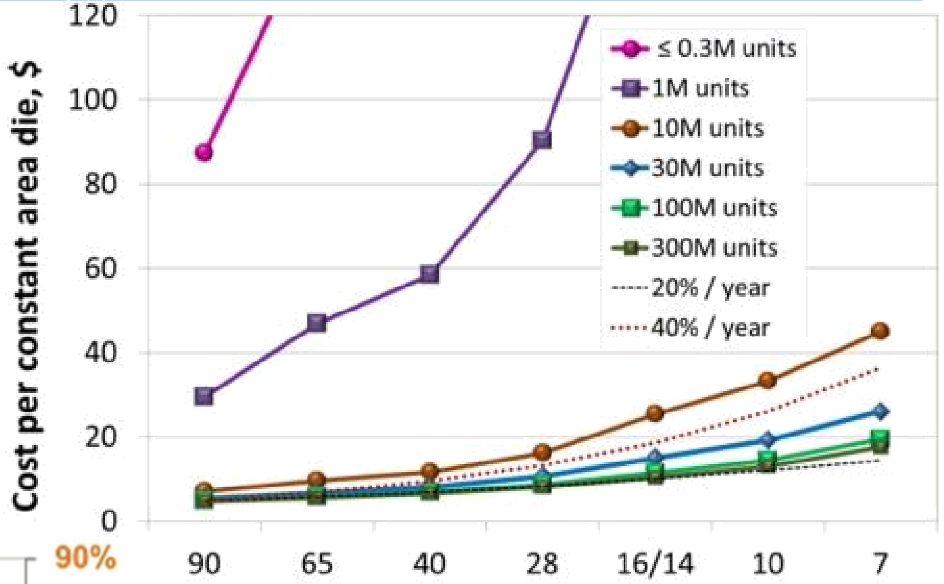
Moore's Law

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

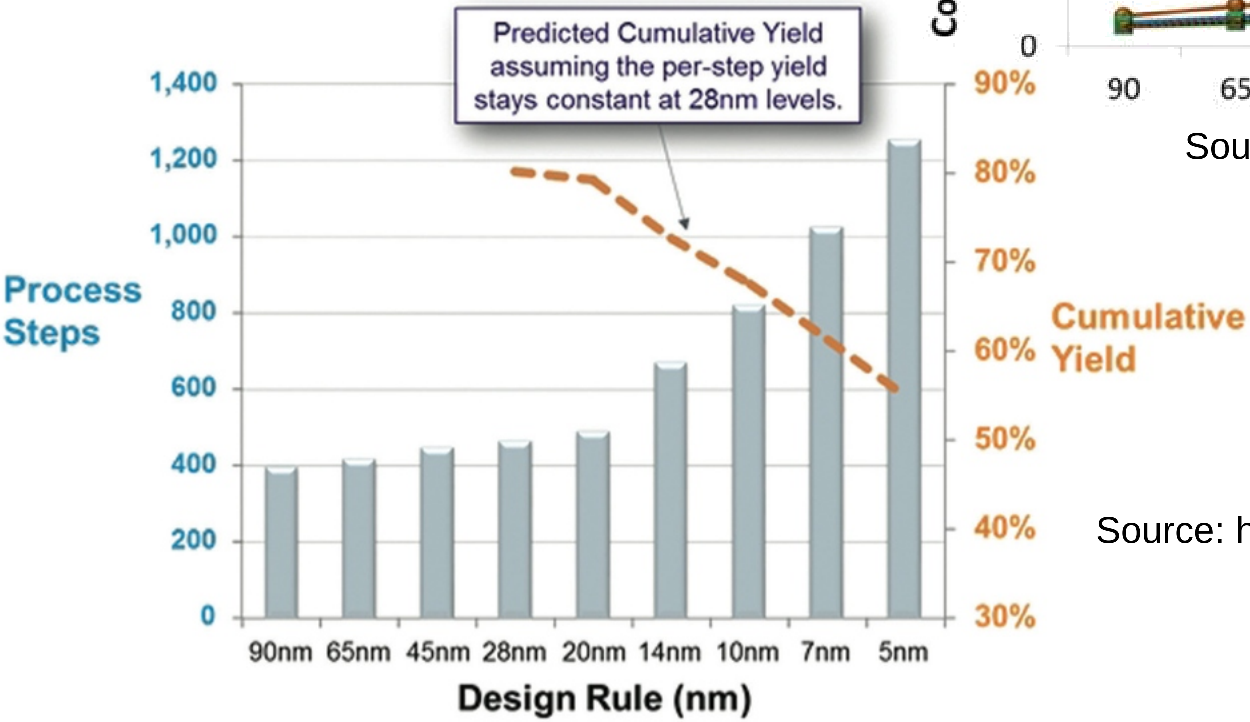


The cost of production is growing with decreasing design rule

Moore's Law will be stopped by cost…

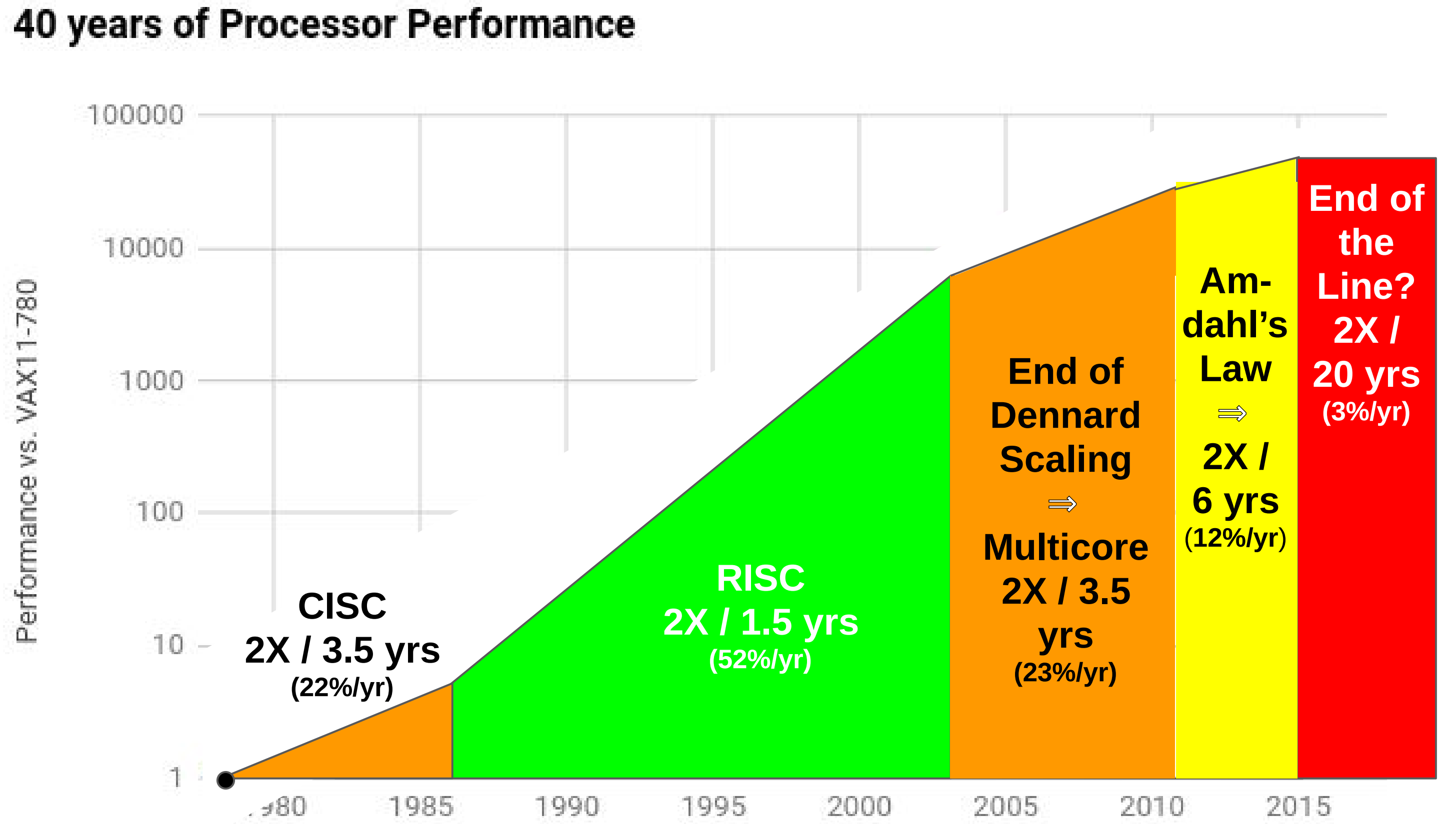


Source: http://www.eetimes.com/



Source: http://electroiq.com/

End of Growth of Single Program Speed?



Based on SPECintCPU. Source: John Hennessy and David Patterson, Computer Architecture: A Quantitative Approach, 6/e. 2018

Processors Architectures Development in a Glimpse

- 1960 – IBM incompatible families → IBM System/360 – one ISA to rule them all,

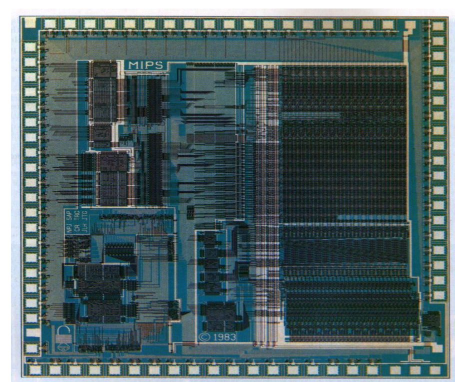
Model	M30	M40	M50	M65
Datapath width	8 bits	16 bits	32 bits	64 bits
Microcode size	4k x 50	4k x 52	2.75k x 85	2.75k x 87
Clock cycle time (ROM)	750 ns	625 ns	500 ns	200 ns
Main memory cycle time	1500 ns	2500 ns	2000 ns	750 ns
Price (1964 $)	$192,000	$216,000	$460,000	$1,080,000
Price (2018 $)	$1,560,000	$1,760,000	$3,720,000	$8,720,000

- 1976 – Writable Control Store, Verification of microprograms, David Patterson Ph.D., UCLA, 1976
- Intel iAPX 432: Most ambitious 1970s micro, started in 1975 – 32-bit capability-based object-oriented architecture, Severe performance, complexity (multiple chips), and usability problems; announced 1981
- Intel 8086 (1978, 8MHz, 29,000 transistors), “Stopgap” 16-bit processor, 52 weeks to new chip, architecture design 3 weeks (10 person weeks) assembly-compatible with 8 bit 8080, further i80286 16-bit introduced some iAPX 432 lapses, i386 paging

Source: A New Golden Age for Computer Architecture with prof. Patterson permission

CISC and RISC

- IBM PC 1981 picks Intel 8088 for 8-bit bus (and Motorola 68000 was out of main business)
- Use SRAM for instruction cache of user-visible instructions
- Use simple ISA – Instructions as simple as microinstructions, but not as wide, Compiled code only used a few CISC instructions anyways, Enable pipelined implementations
- Chaitin's register allocation scheme benefits load-store ISAs
- Berkeley (RISC I, II → SPARC) & Stanford RISC Chips (MIPS)



Stanford MIPS (1983) contains 25,000 transistors, was fabbed in 3 µm &4 µm NMOS, ran at 4 MHz (3 µm), and size is 50 mm2 (4 µm) (Microprocessor without Interlocked Pipeline Stages)

Source: A New Golden Age for Computer Architecture with prof. Patterson permission

CISC and RISC – PC and Post PC Era

- CISC executes fewer instructions per program (≈ 3/4X instructions), but many more clock cycles per instruction (≈ 6X CPI)

  ⇒ RISC ≈ 4X faster than CISC

PC Era
- Hardware translates x86 instructions into internal RISC Instructions (Compiler vs Interpreter)
- Then use any RISC technique inside MPU
- > 350M / year !
- x86 ISA eventually dominates servers as well as desktops

PostPC Era: Client/Cloud
- IP in SoC vs. MPU
- Value die area, energy as much as performance
- > 20B total / year in 2017
- 99% Processors today are RISC
- Marketplace settles debate

- Alternative, Intel Itanium VLIW, 2002 instead 1997
- "*The Itanium approach...was supposed to be so terrific –until it turned out that the wished-for compilers were basically impossible to write.*" - Donald Knuth, Stanford

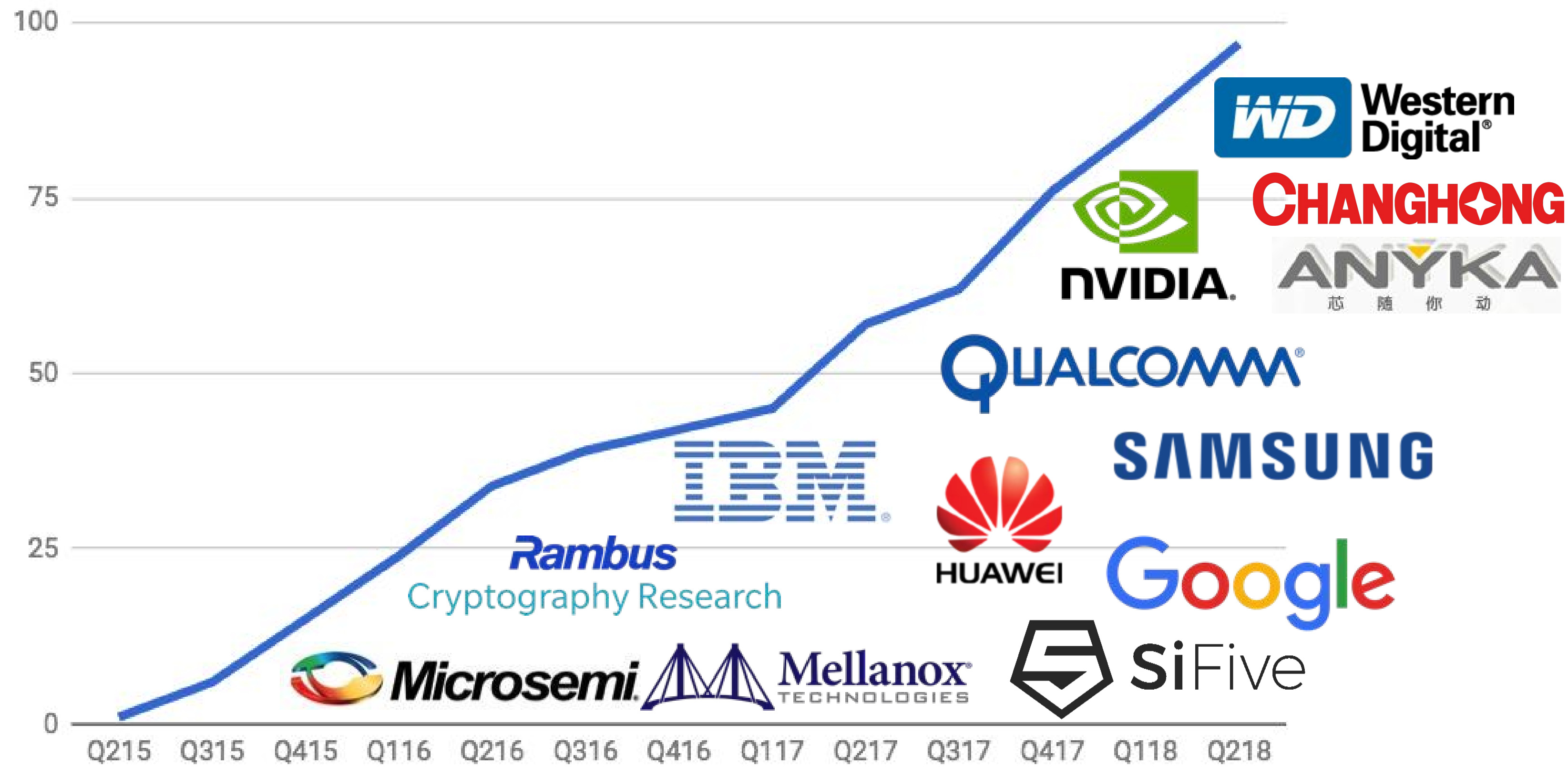
Source: A New Golden Age for Computer Architecture with prof. Patterson permission

RISC-V

- ARM, MIPS, SPARC, PowerPC – Commercialization and extensions results in too complex CPUs again, with license and patents preventing even original investors to use real/actual implementations in silicon to be used for education and research
- Krste Asanovic and other prof. Patterson's students initiated development of new architecture (start of 2010), initial estimate to design architecture 3 months, but 3 years
- Simple, Clean-slate design (25 years later, so can learn from mistakes of predecessors, Avoids µarchitecture or technology-dependent features), Modular, Supports specialization, Community designed
- A few base integer ISAs (RV32E, RV32I, RV64I)
- Standard extensions (M: Integer multiply/divide, A: Atomic memory operations, F/D: Single/Double-precision Fl-point, C: Compressed Instructions (<x86), V: Vector Extension for DLP (>SIMD**))
- Domain Specific Architectures (DSAs)

Source: A New Golden Age for Computer Architecture with prof. Patterson permission

RISC-V Foundation Members since 2015



Open Architecture Goal

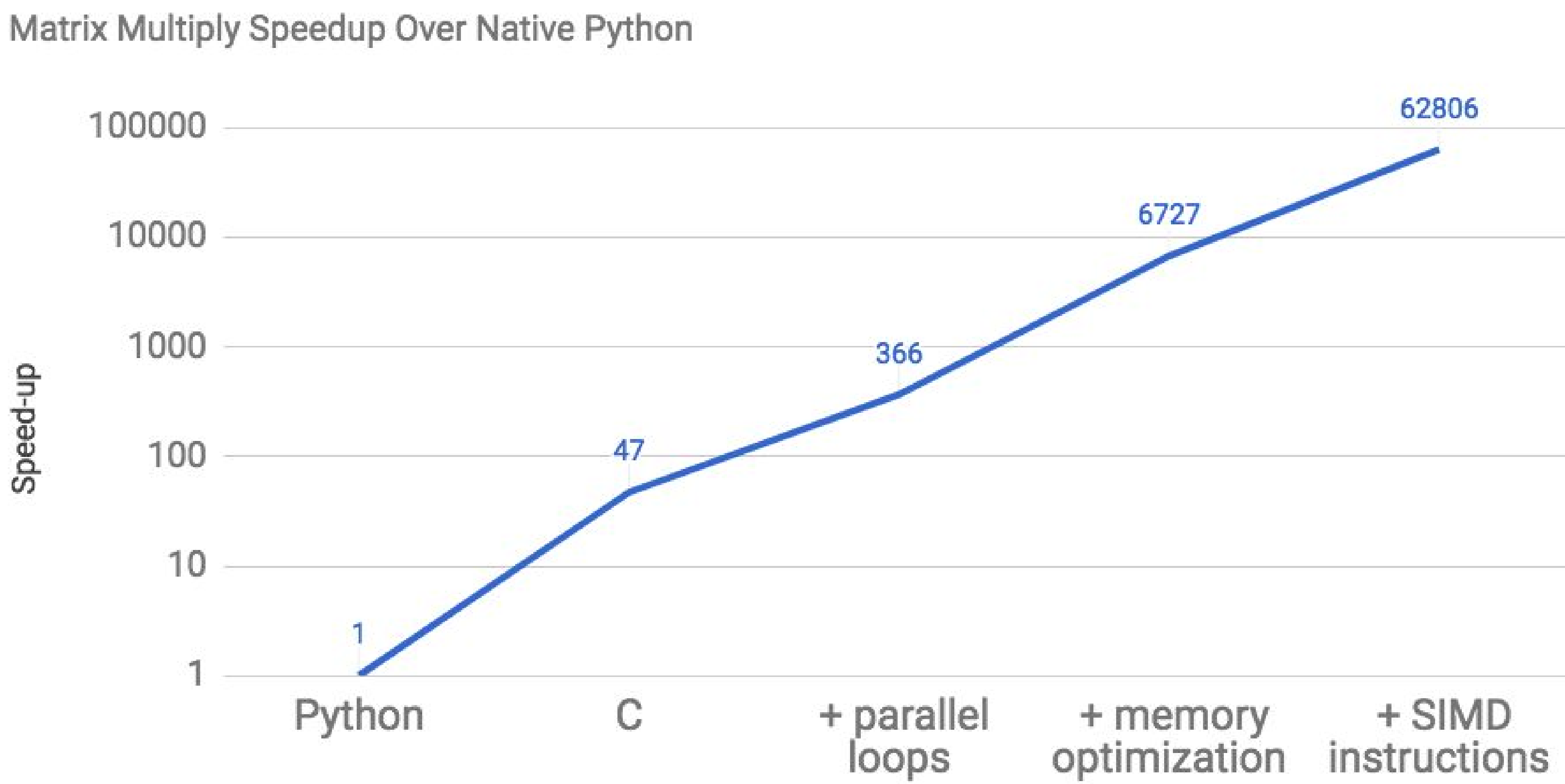
Create industry-standard open ISAs for all computing devices

“Linux for processors”

Source: A New Golden Age for Computer Architecture with prof. Patterson permission

What’s the Opportunity?

Matrix Multiply: relative speedup to a Python version (18 core Intel)



Source: “There’s Plenty of Room at the Top,” Leiserson, et. al., *to appear*.

Source: A New Golden Age for Computer Architecture with prof. Patterson permission

Domain Specific Architectures (DSAs)

- Achieve higher efficiency by tailoring the architecture to characteristics of the domain
- Not one application, but a domain of applications

  -Different from strict ASIC
- Requires more domain-specific knowledge then general purpose processors need
- Examples: (Neural network processors for machine learning, GPUs for graphics, virtual reality)
- Programmable network switches and interfaces
- More effective parallelism for a specific domain: (SIMD vs. MIMD, VLIW vs. Speculative, out-of-order)
- More effective use of memory bandwidth (User controlled versus caches)
- Eliminate unneeded accuracy (IEEE replaced by lower precision FP, 32-64 bit bit integers to 8-16 bit integers)
- Domain specific programming language

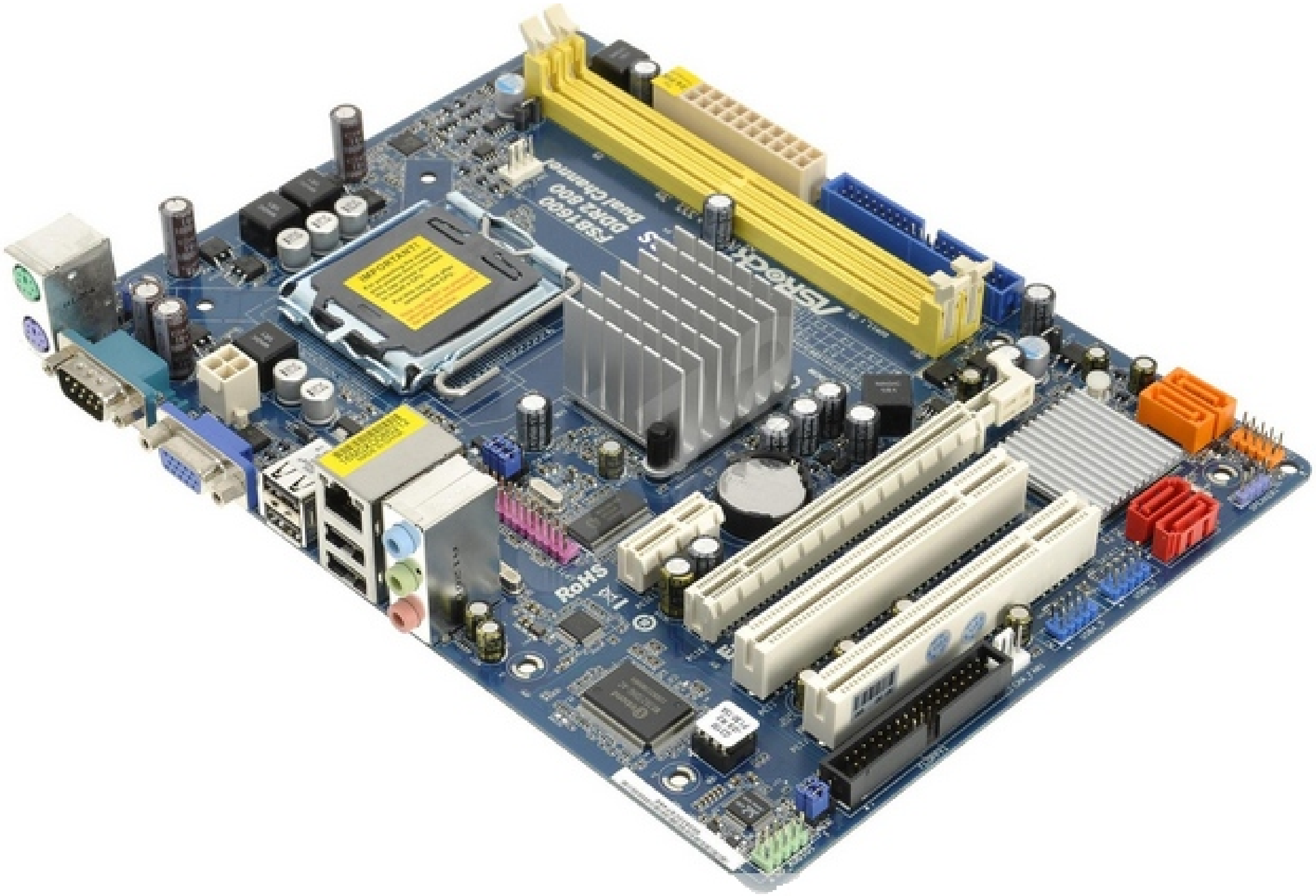
Source: A New Golden Age for Computer Architecture with prof. Patterson permission

DSL/DSA Summary

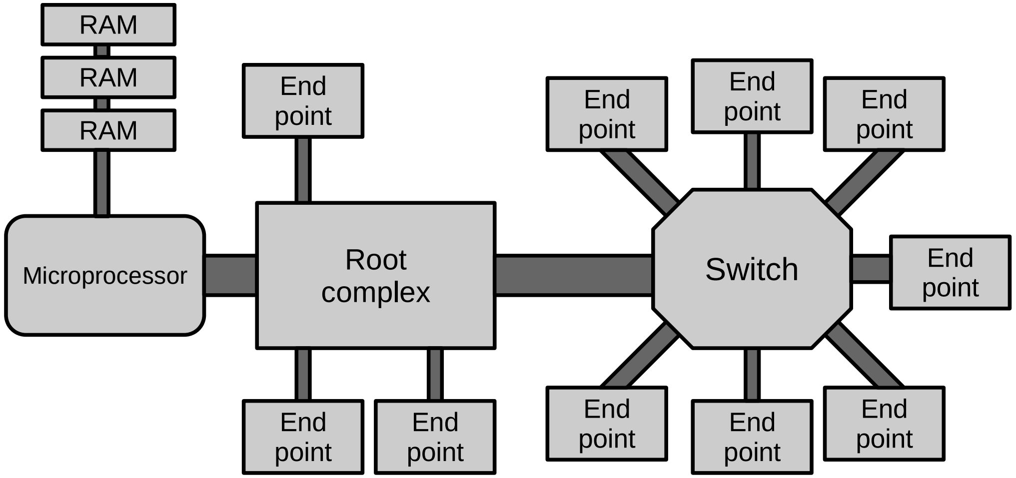
- Lots of opportunities
- But, new approach to computer architecture is needed
- The Renaissance computer architecture team is vertically integrated. Understands:
  - Applications
  - DSLs and related compiler technology
  - Principles of architecture
  - Implementation technology
- Everything old is new again!
- Open Architectures
  - Why open source compilers and operating systems but not ISAa?
  - What if there were free and open ISAs we could use for everything?

Source: A New Golden Age for Computer Architecture with prof. Patterson permission

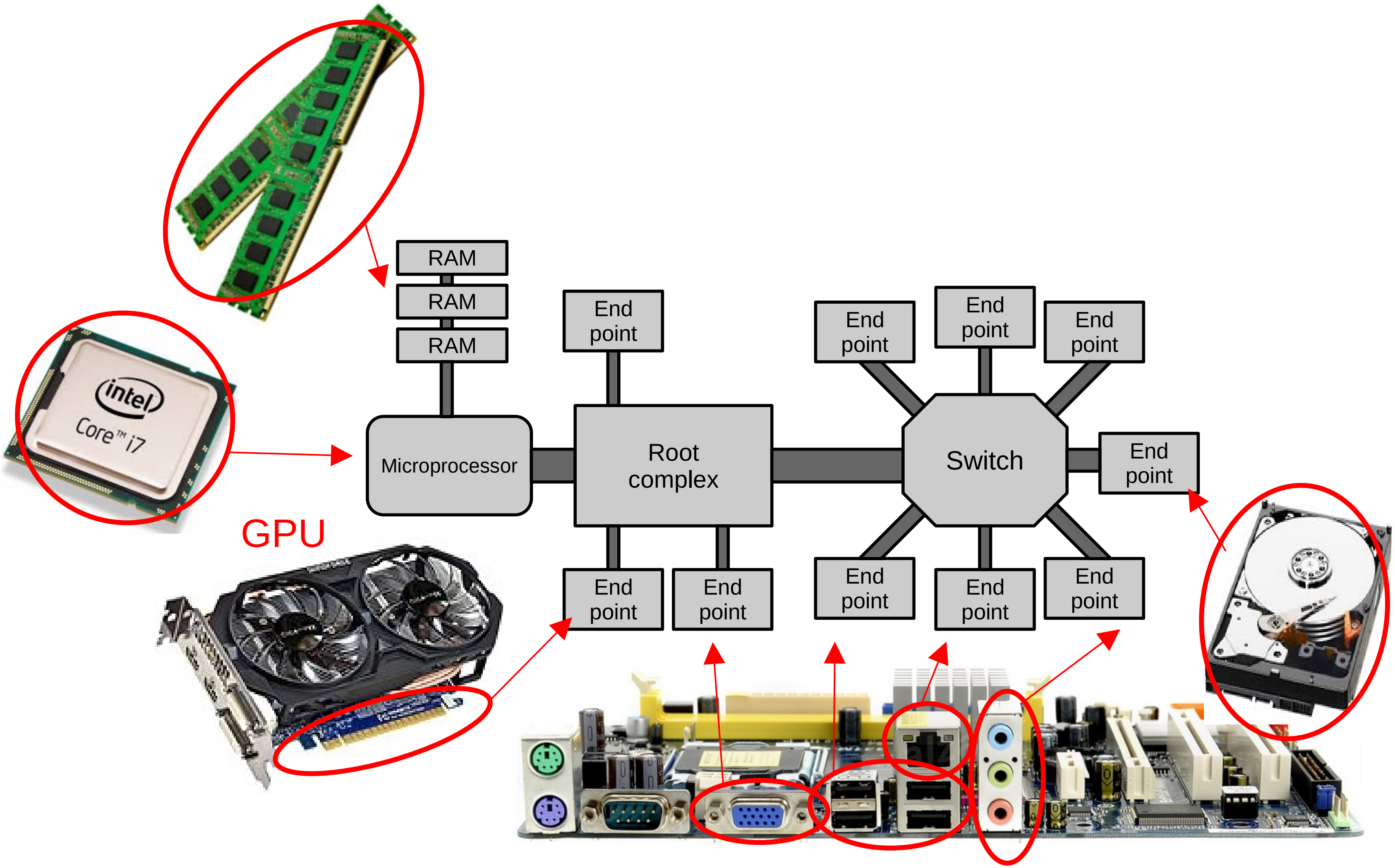
Today PC Computer Base Platform – Motherboard



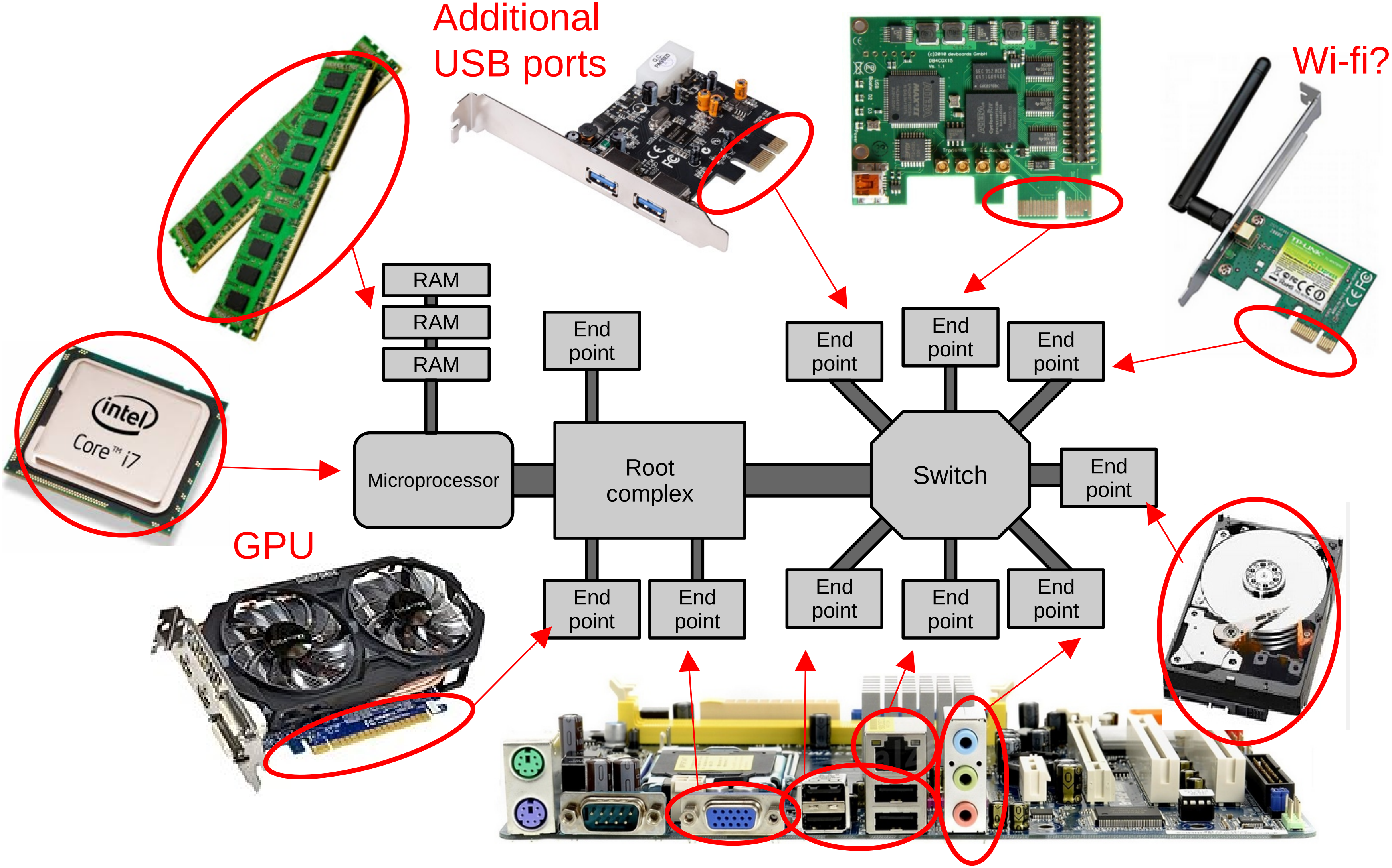
Block Diagram of Components Interconnection



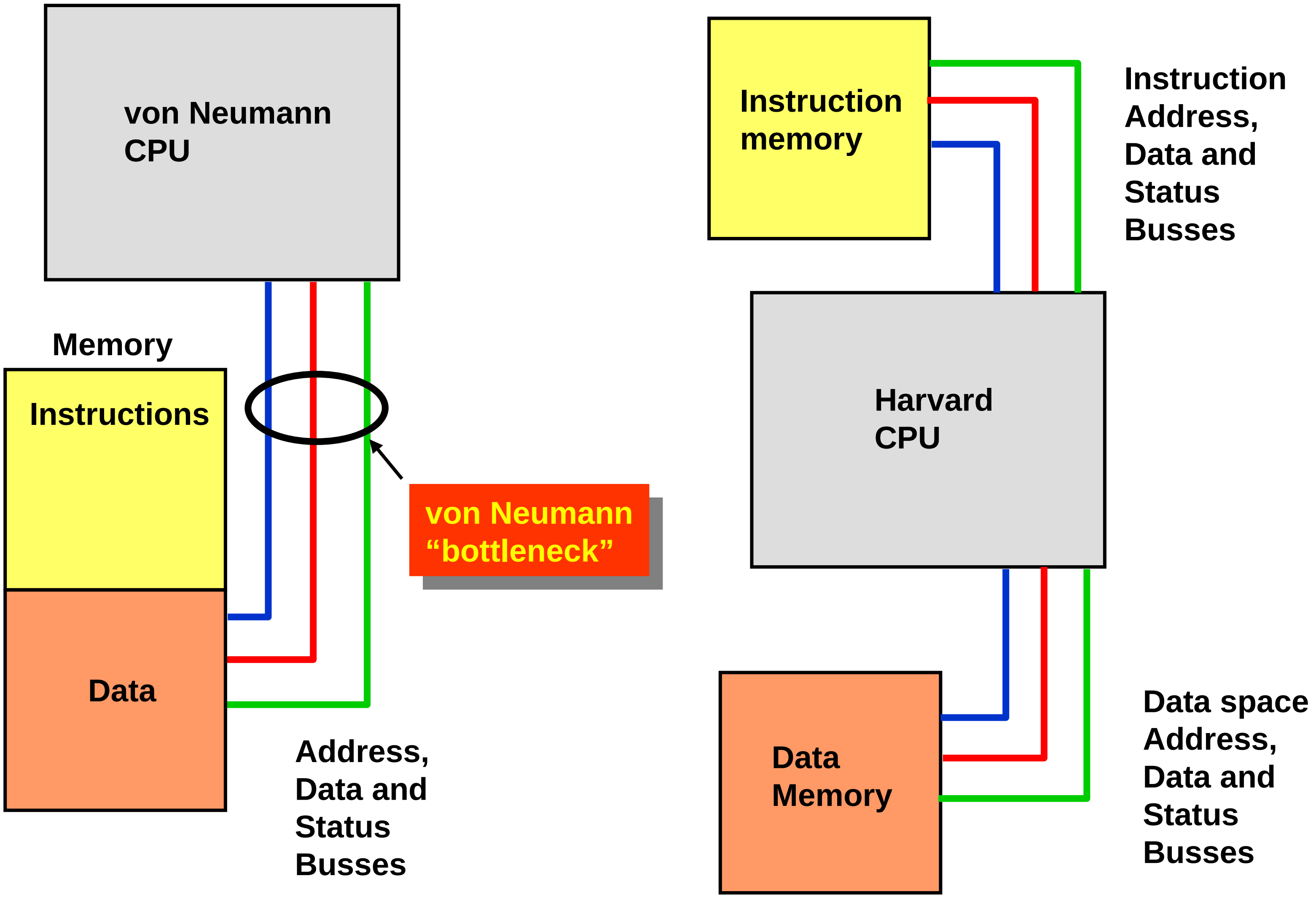
Block Diagram of Components Interconnection



Block Diagram of Components Interconnection



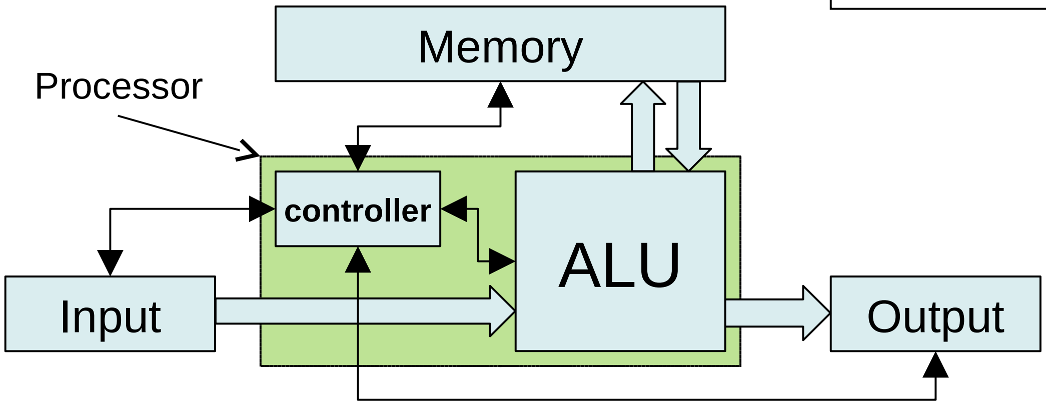
Von Neumann and Harvard Architectures



[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

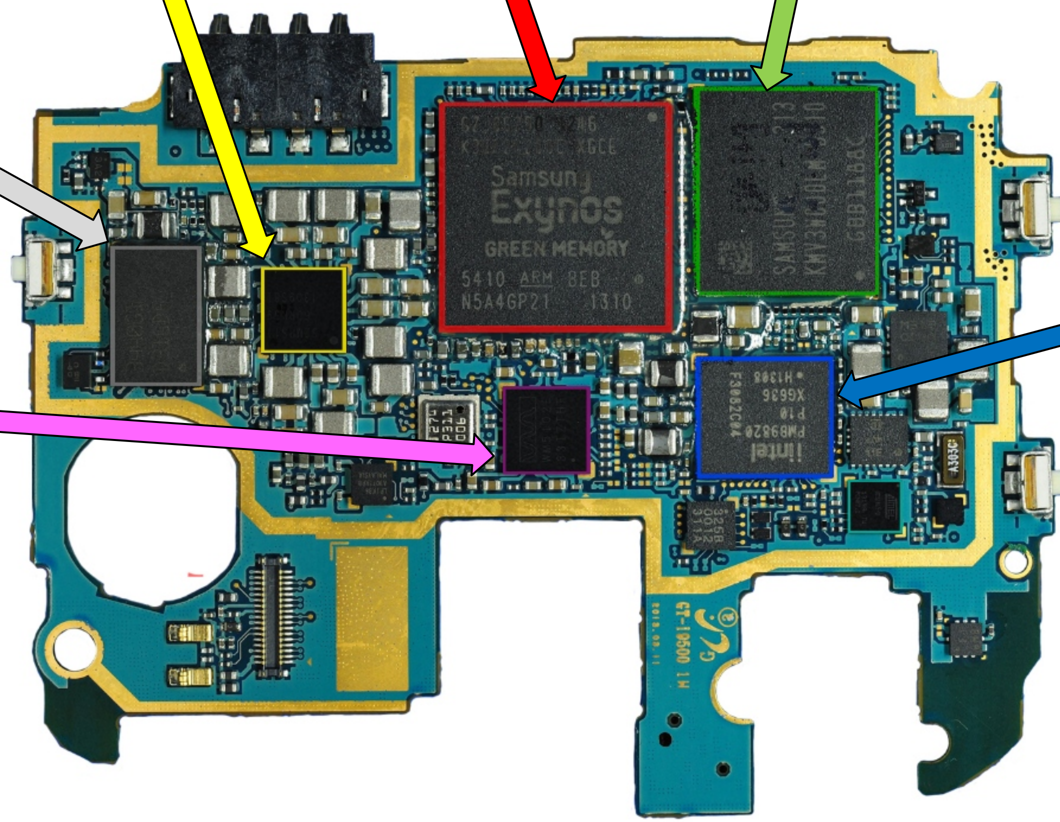
Exynos 5410 (**8-core CPU + 2GB DRAM**)

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

Wi-fi (broadcom BCM4335)

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

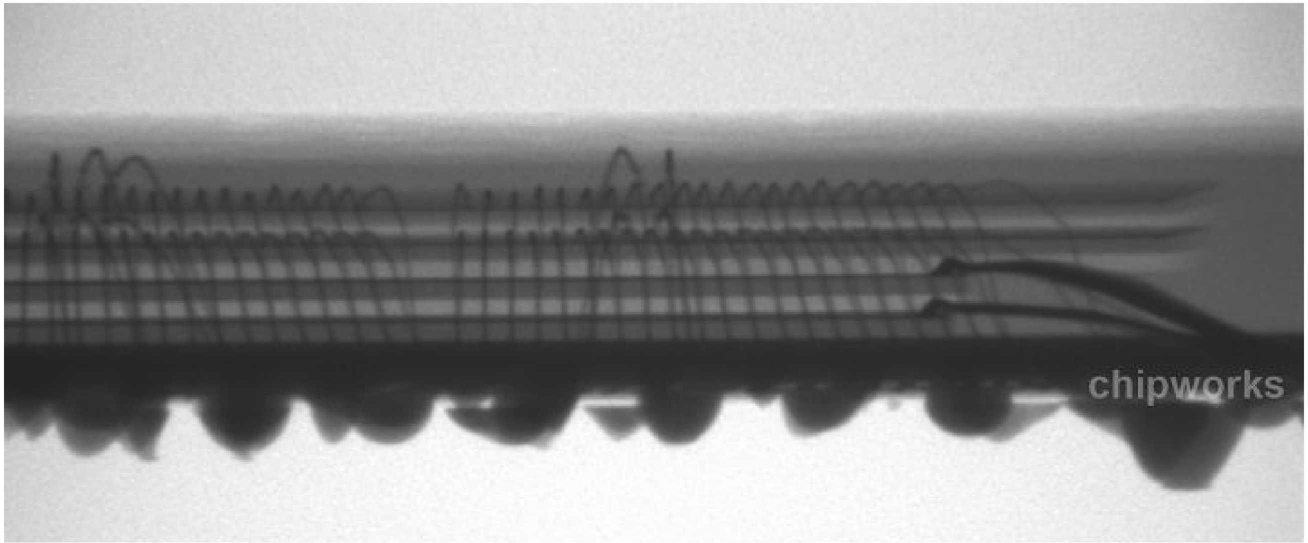
DSP processor for voice and audio codec



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]*

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

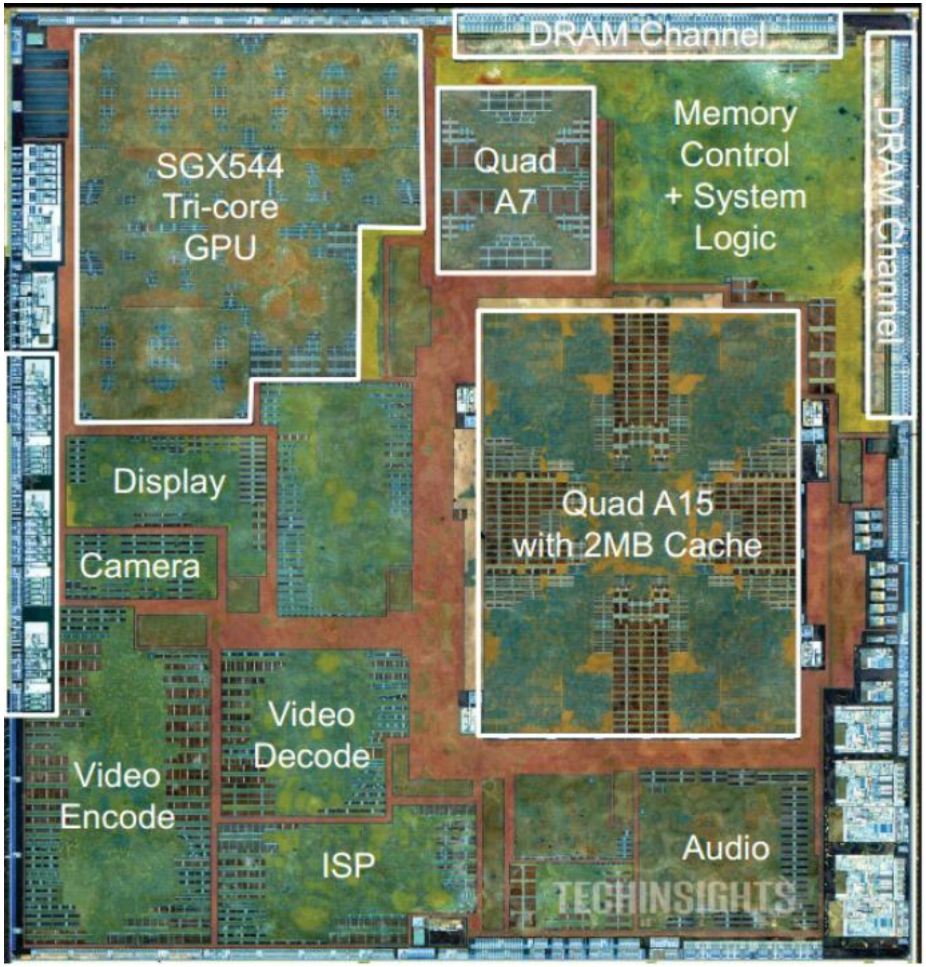
Samsung Galaxy S4 inside

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

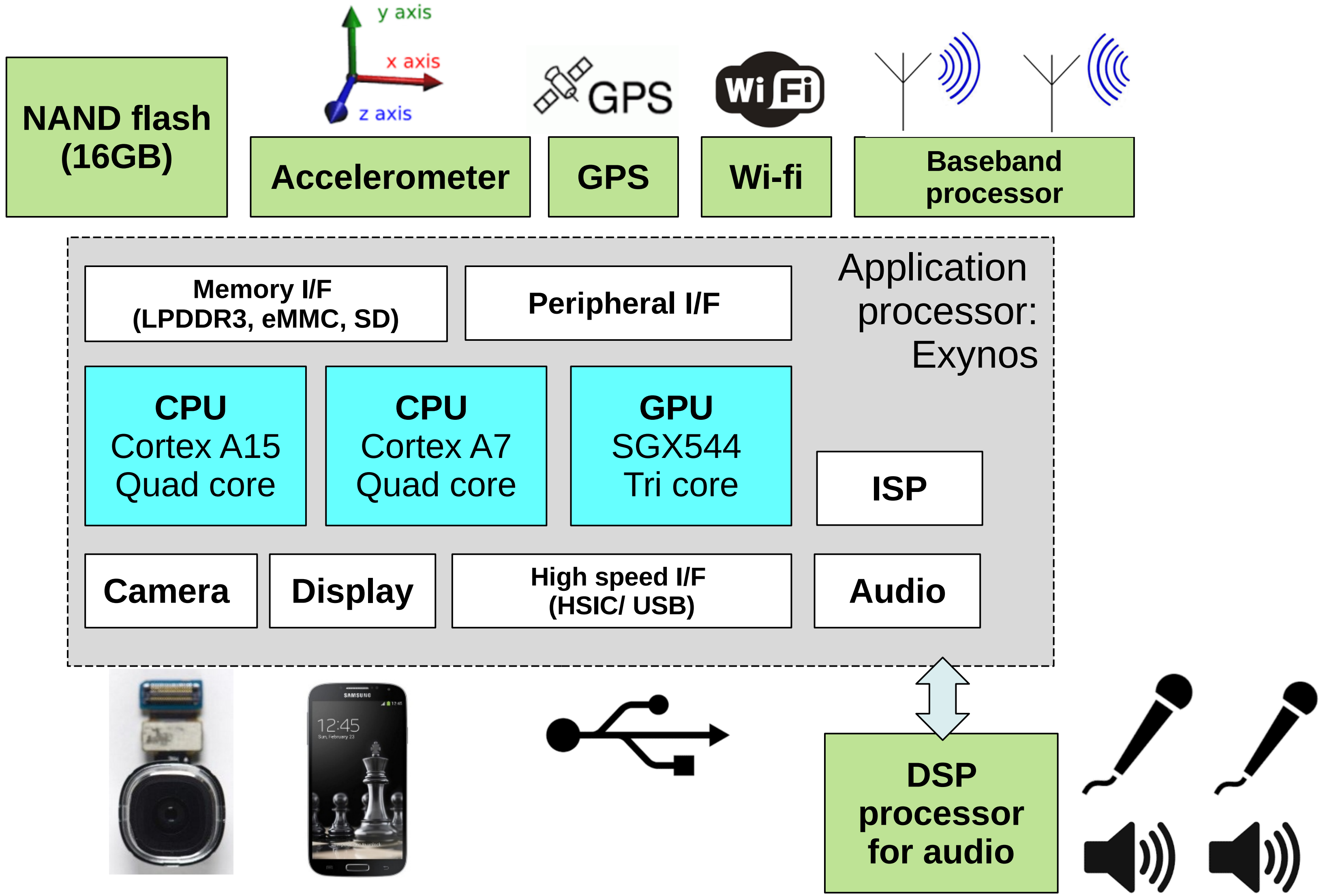
Chip Exynos 5410



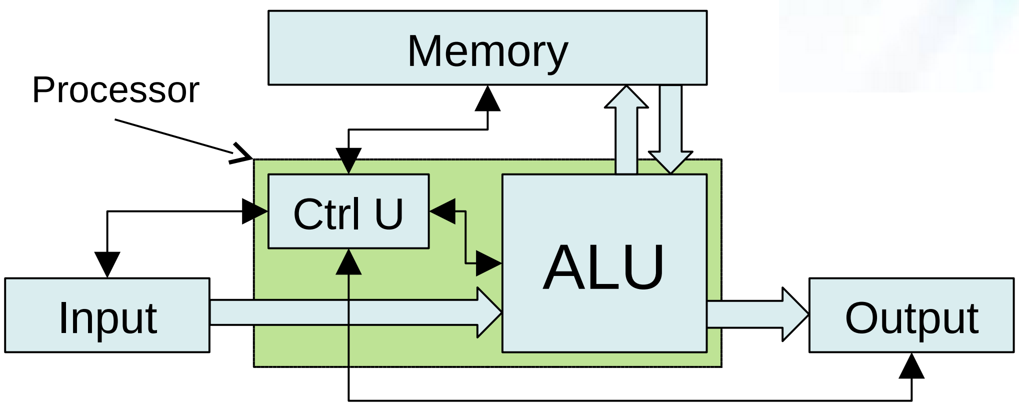
- Note the different sizes of 4 cores A7 and 4 cores A15
- On the chip, other components are integrated outside the processor: the GPU, Video coder and decoder, and more. This is SoC (System on Chip)

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



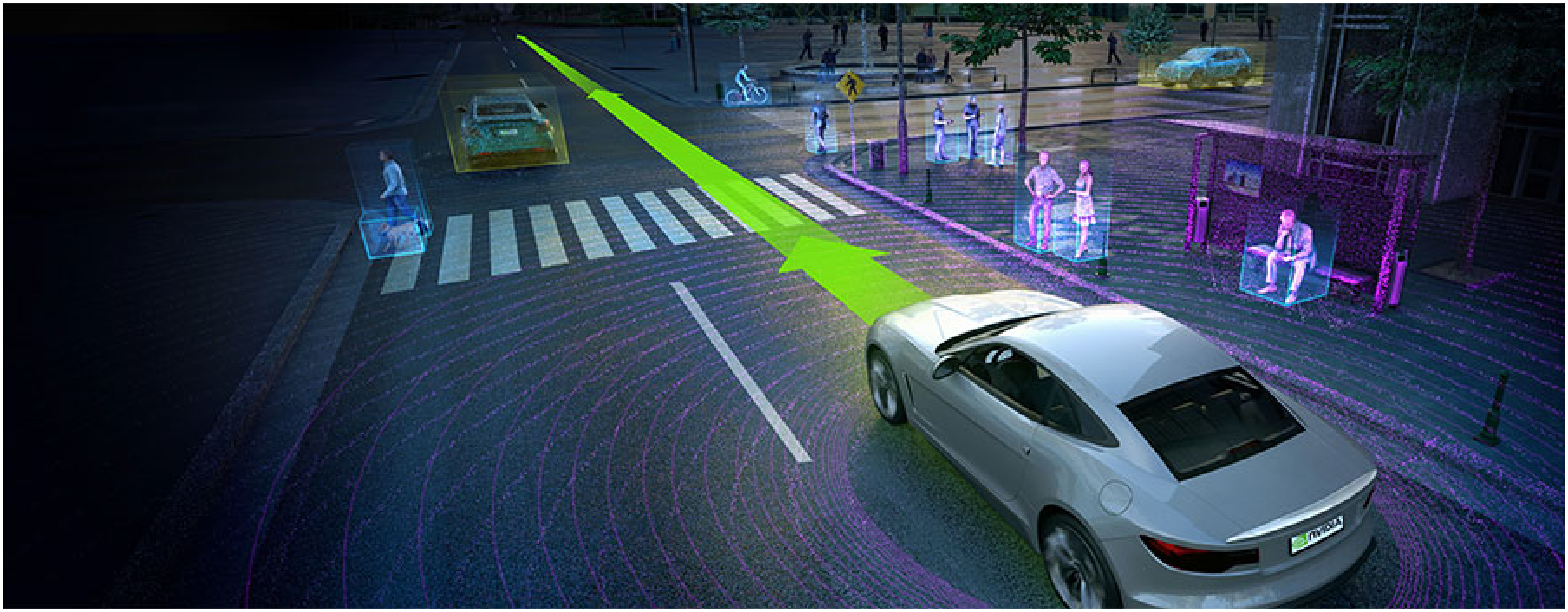
Common concept



- 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

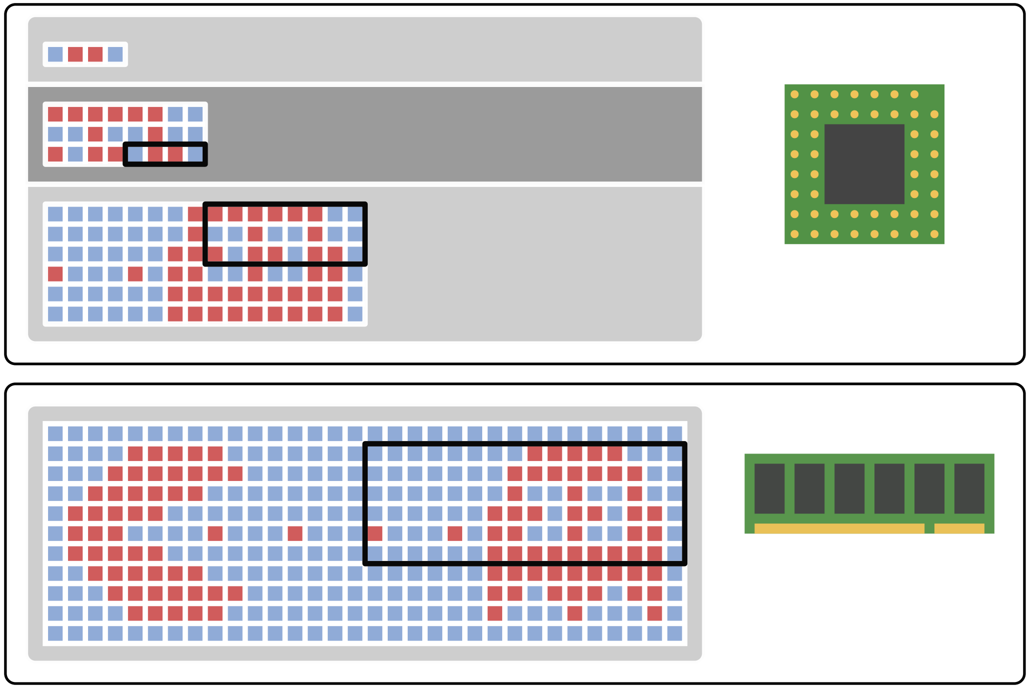
- Many artificial intelligence tasks are based on deep neural networks (deep neural networks)

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, compiler flags -03), GPU results Joela Matějka, Department of Control Engineering, FEE, CTU https://dce.fel.cvut.cz/
- How to speedup?

Optimize Memory Accesses

- Algorithm modification with respect to memory hierarchy
- Data from the (buffer) memory near the processor can be obtained faster (but fast memory is small in size)



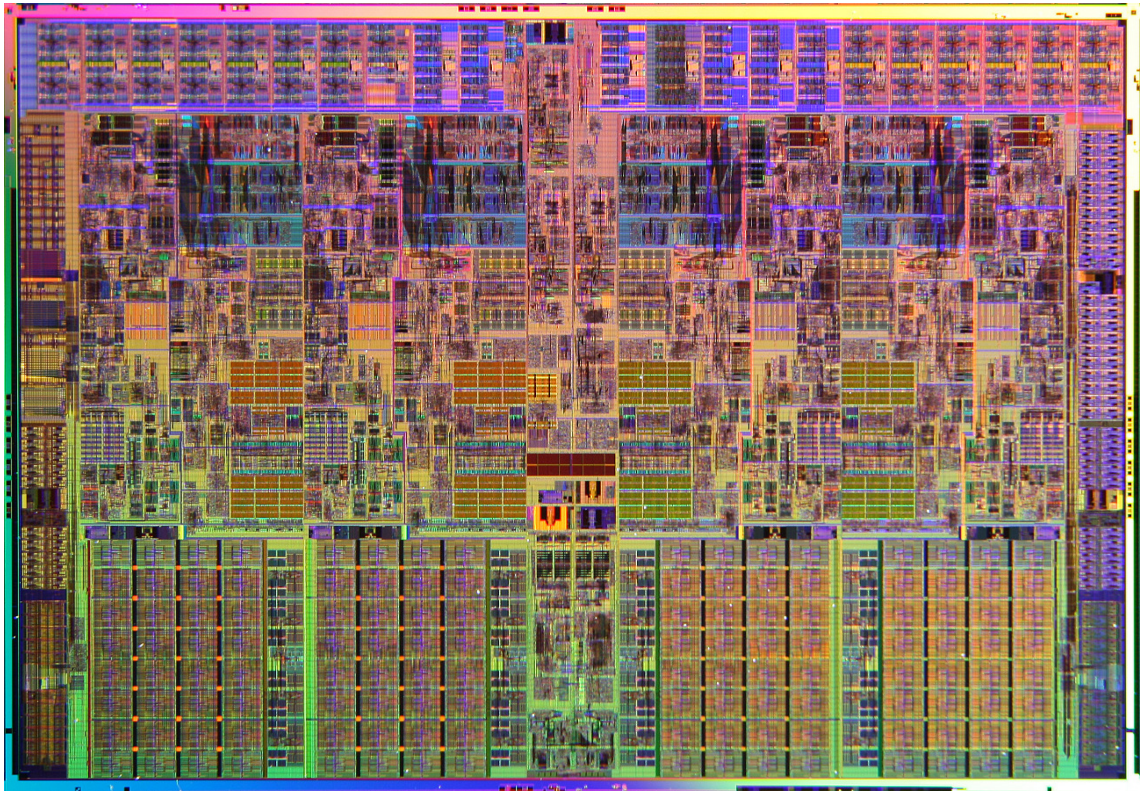
Prediction of jumps / accesses to memory

- In order to increase average performance, the execution of instructions is divided into several phases => the need to read several instructions / data in advance
- Every condition (if, loop) means a possible jump - poor prediction is expensive
- It is good to have an idea of how the predictions work and what alternatives there are on the CPU / HW. (Eg vector / multimedia inst.)

Source: https://commons.wikimedia.org/wiki/File:Plektita_trakforko_14.jpeg

Parallelization - Multicore Processor

- Synchronization requirements
- Interconnection and communication possibilities between processors
- Transfers between memory levels are very expensive
- Improper sharing/access form more cores results in slower code than on a single CPU



Intel Nehalem Processor, Original Core i7

Source: http://download.intel.com/pressroom/kits/corei7/images/Nehalem_Die_Shot_3.jpg

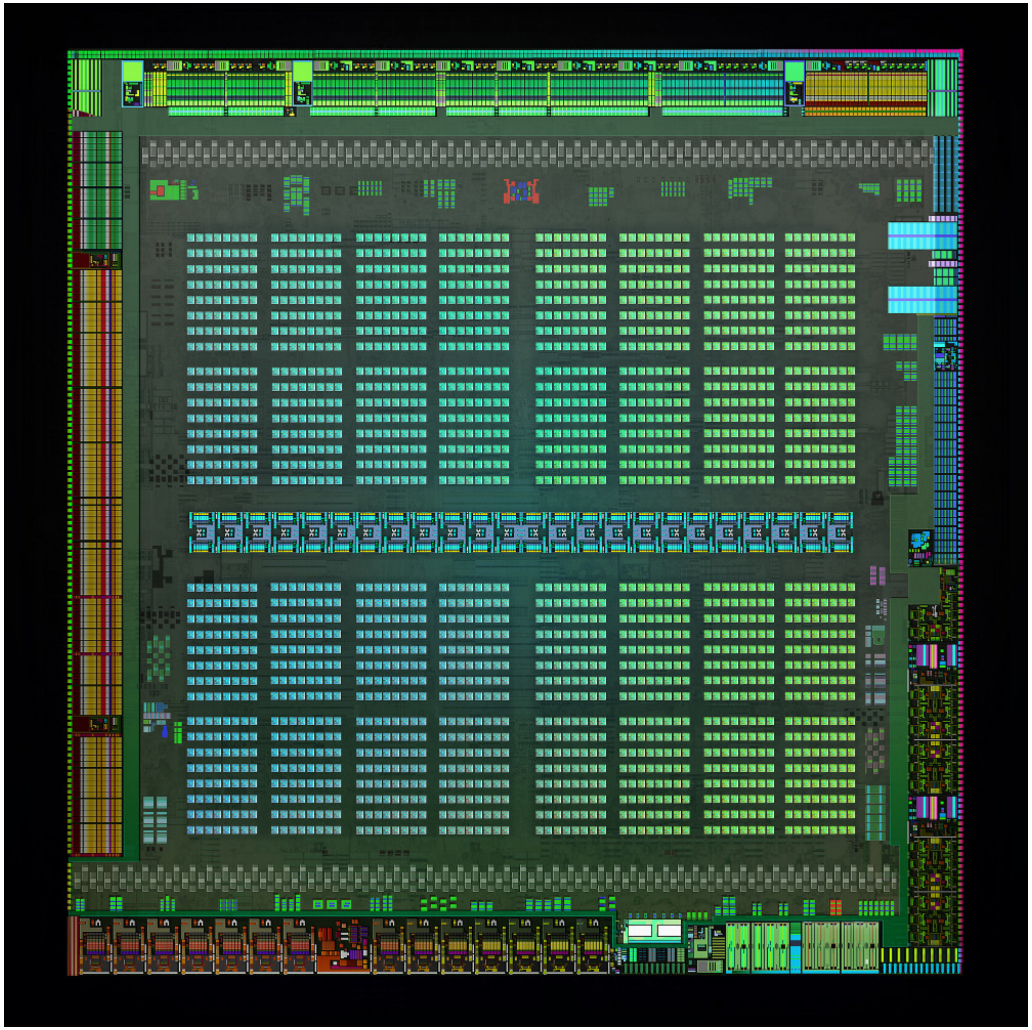
Computing Coprocessors - GPU

- Multi-core processor (hundreds)
- Some units and bclocks shared
- For effective use it is necessary to know the basic hardware features

Source: https://devblogs.nvidia.com/parallelforall/inside-pascal/

GPU – Maxwell

- GM204
- 5200 milins trasistors
- 398 mm^2
- PCIe 3.0 x16
- 2048 computation units
- 4096 MB
- 1126 MHz
- 7010 MT/s
- 72.1 GP/s
- 144 GT/s
- 224 GB/s

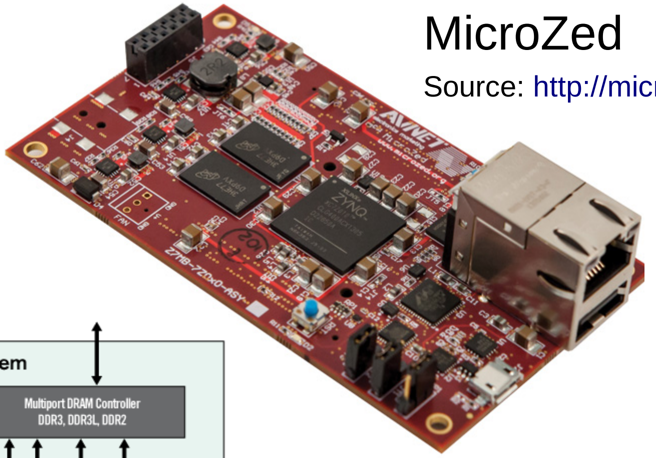


Source: http://www.anandtech.com/show/8526/nvidia-geforce-gtx-980-review/3

FPGA – design/prototyping of own hardware

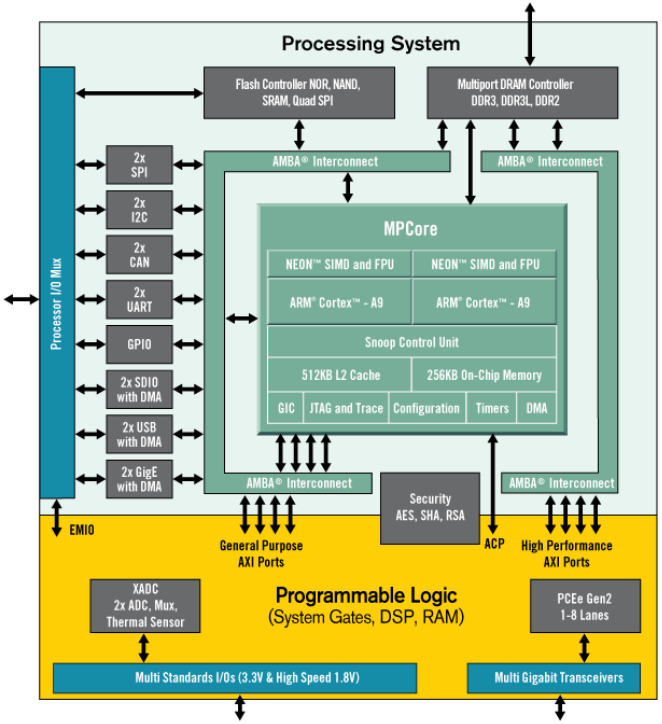
- Programmable logic arrays
- Well suited for effective implementaion of some digital signal manipulation (filters – images, video or audio, FFT analysis, custom CPU architecture…)
- Programmer interconnects blcoks available on the chip
- Zynq 7000 FPGA – two ARM cores equipped by FPGA – fast and simple access to FPGA/peripherals from own program
- (the platform is used for your seminaries but you will use only design prepared by us, the FPGA programming/logic design is topic for more advance couses)

Xilinx Zynq 7000 a MicroZed APO

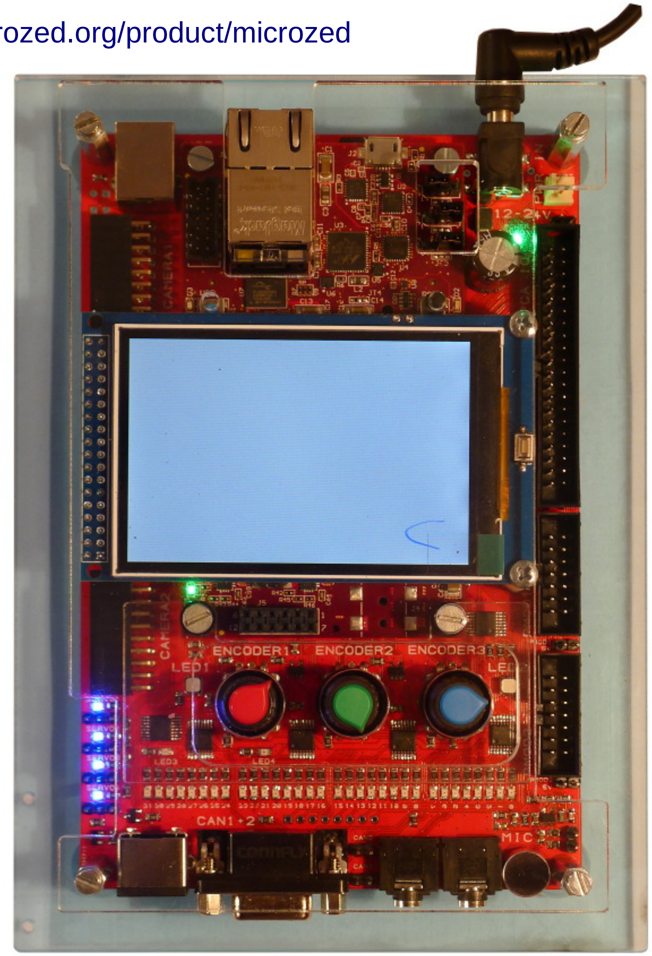


MicroZed

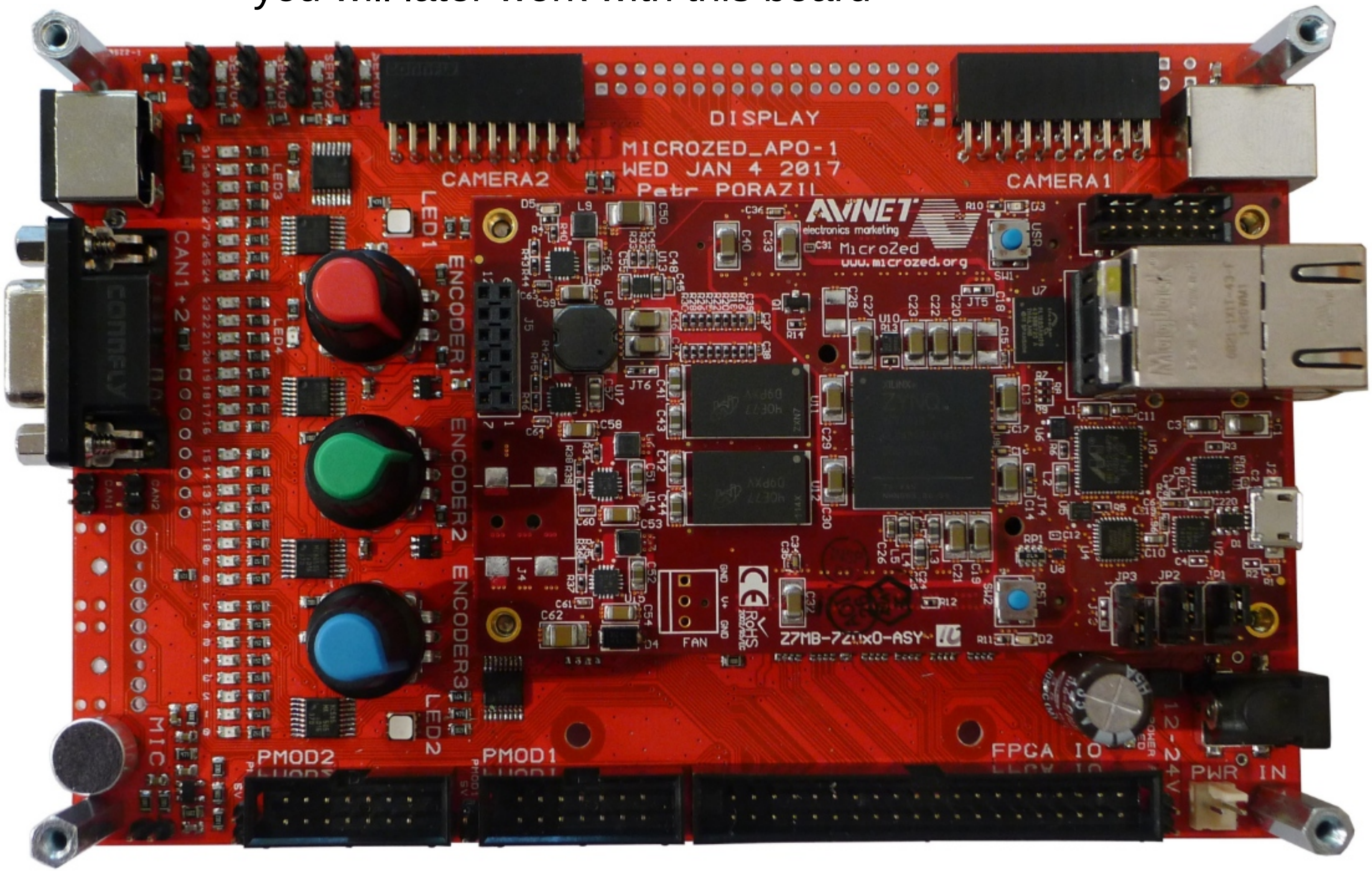
Source: http://microzed.org/product/microzed



Source: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html



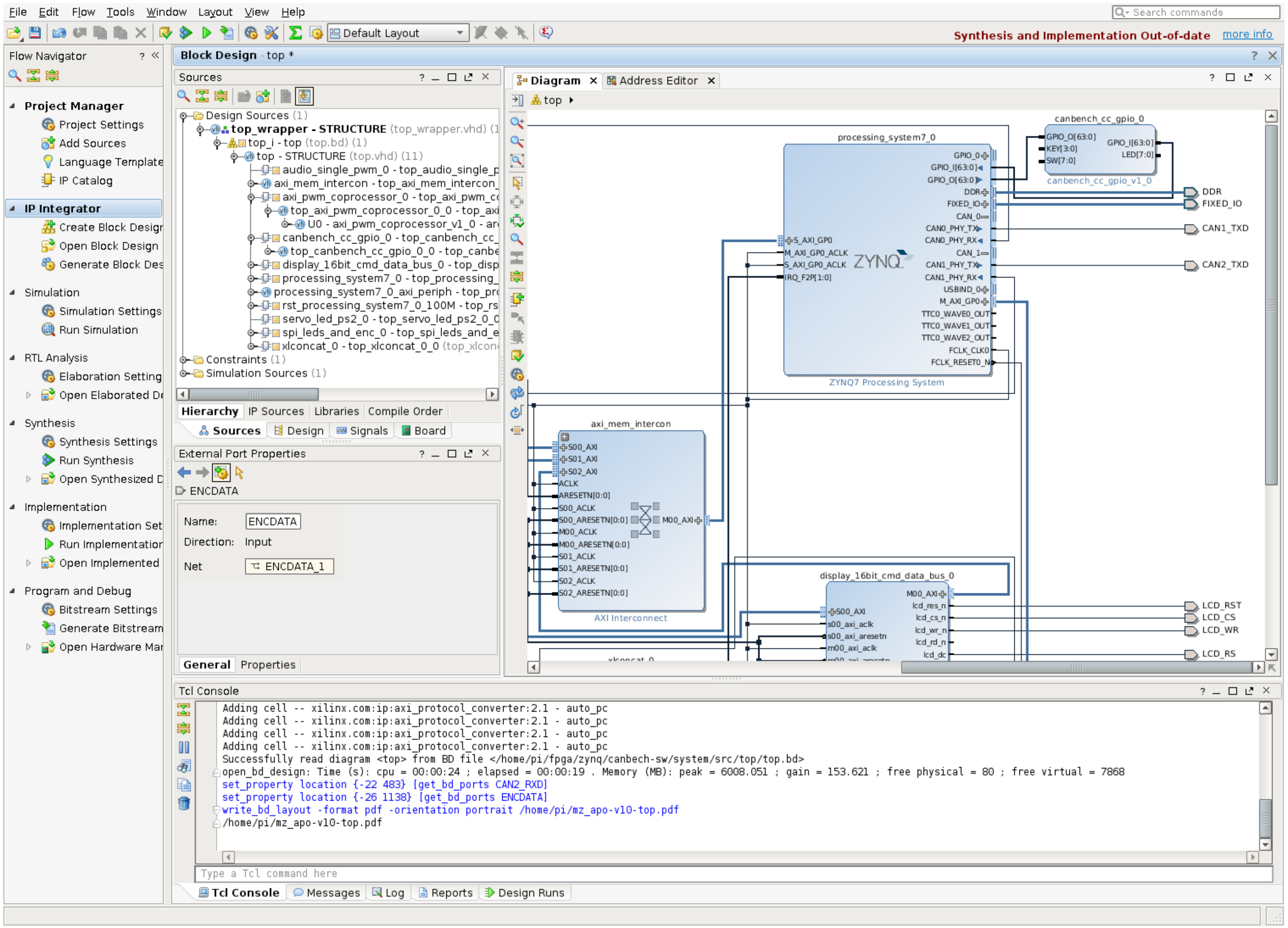
Source: https://cw.fel.cvut.cz/wiki/courses/b35apo/start you will later work with this board



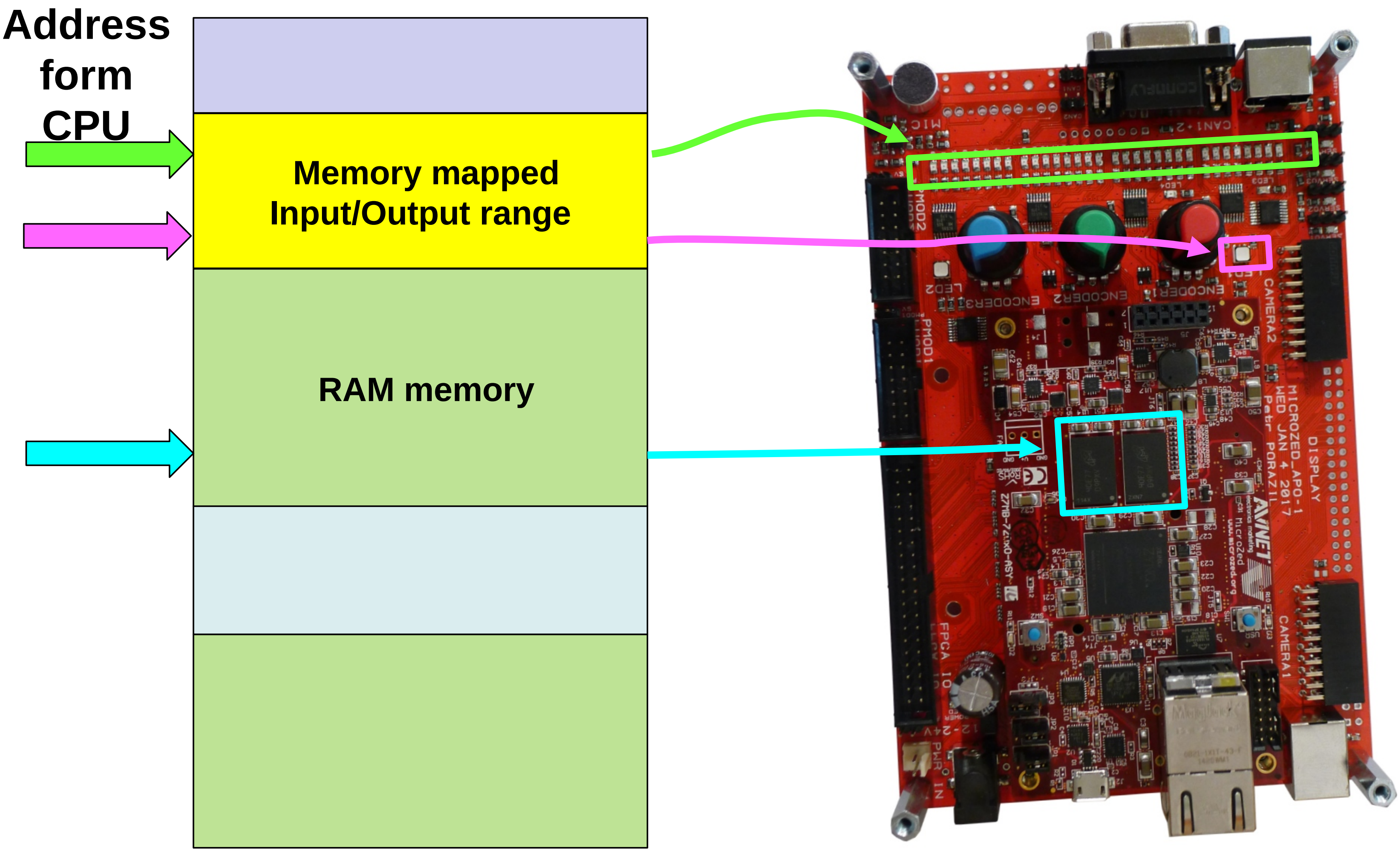
MZ_APO – Features

- The core chip: Zynq-7000 All Programmable SoC
- Typ: Z-7010, device XC7Z010
- CPU: Dual ARM® Cortex™-A9 MPCore™ @ 866 MHz (NEON™ & Single / Double Precision Floating Point) 2x L1 32+32 kB, L2 512 KB
- FPGA: 28K Logic Cells (~430K ASIC logic gates, 35 kbit)
- Computational capability of FPGA DSP blocks: 100 GMACs
- Memory for FPGA design: 240 KB
- Memory on MicroZed board: 1GB
- Operating system: GNU/Linux
  - GNU LIBC (libc6) 2.28-10
  - Kernel: Linux 4.19.59-rt22-00005-gedf9096397ae
  - Distribution: Debian 10 (Buster)

MZ_APO – Logic Design Developed in Xilinx Vivado



MZ_APO – Peripherals in Physical Address Space



GNU/Linux operating system – from tiny gadgets ...

Linux – from Tiny to Supercomputers

- TOP500 https://www.top500.org/ (https://en.wikipedia.org/wiki/TOP500)
- Actual top one: Summit supercomputer – IBM AC922
- June 2018, US Oak Ridge National Laboratory (ORNL),
- 200 PetaFLOPS, 4600 “nodes”, 2× IBM Power9 CPU +
- 6× Nvidia Volta GV100
- 96 lanes of PCIe 4.0, 400Gb/s
- NVLink 2.0, 100GB/s CPU-to-GPU,
- GPU-to-GPU
- 2TB DDR4-2666 per node
- 1.6 TB NV RAM per node
- 250 PB storage
- POWER9-SO, Global Foundries 14nm FinFET,
  8×109 tran., 17-layer, 24 cores, 96 threads (SMT4)
- 120MB L3 eDRAM (2 CPU 10MB), 256GB/s

Source: http://www.tomshardware.com/

- Other example: SGI SSI (single system image) Linux, 2048 Itanium CPU a 4TiB RAM

Linux Kernel and Open-source

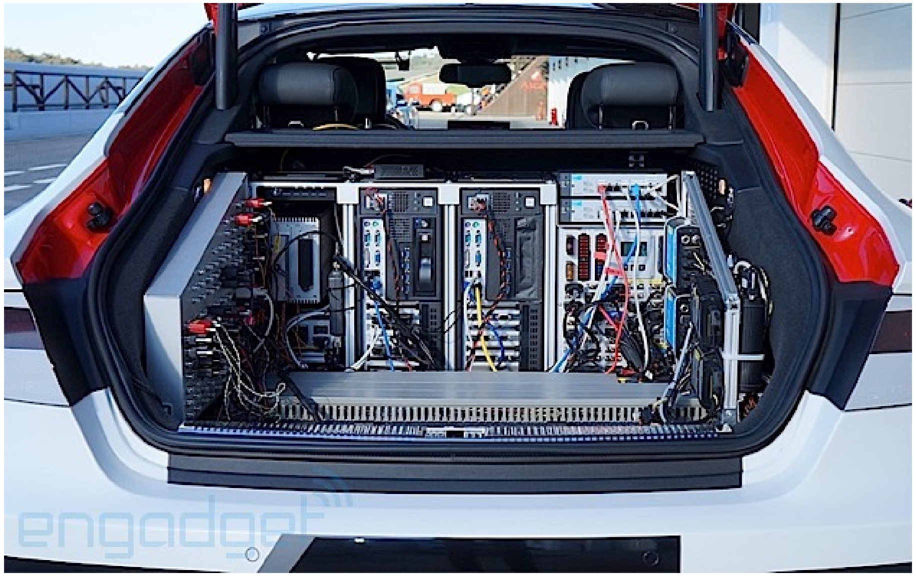
- Linux kernel project
  - 13,500 developers from 2005 year
  - 10,000 lines of code inserted daily
  - 8,000 removed and 1,500 till 1,800 modified
  - GIT source control system
- Many successful open-source projects exists
- Open for joining by everybody
- Google Summer of Code for university students
  - https://developers.google.com/open-source/gsoc/

Zdroj: https://www.theregister.co.uk/2017/02/15/think_different_shut_up_and_work_harder_says_linus_torvalds/

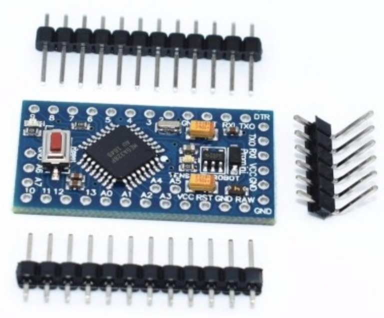
Back to the Motivational Example of Autonomous Driving

The result of a good knowledge of hardware

- Acceleration (in our case 18 × using the same number of cores)
- Reduce the power required
- Energy saving
- Possibility to reduce current solutions
- Using GPUs, we process 40 fps.



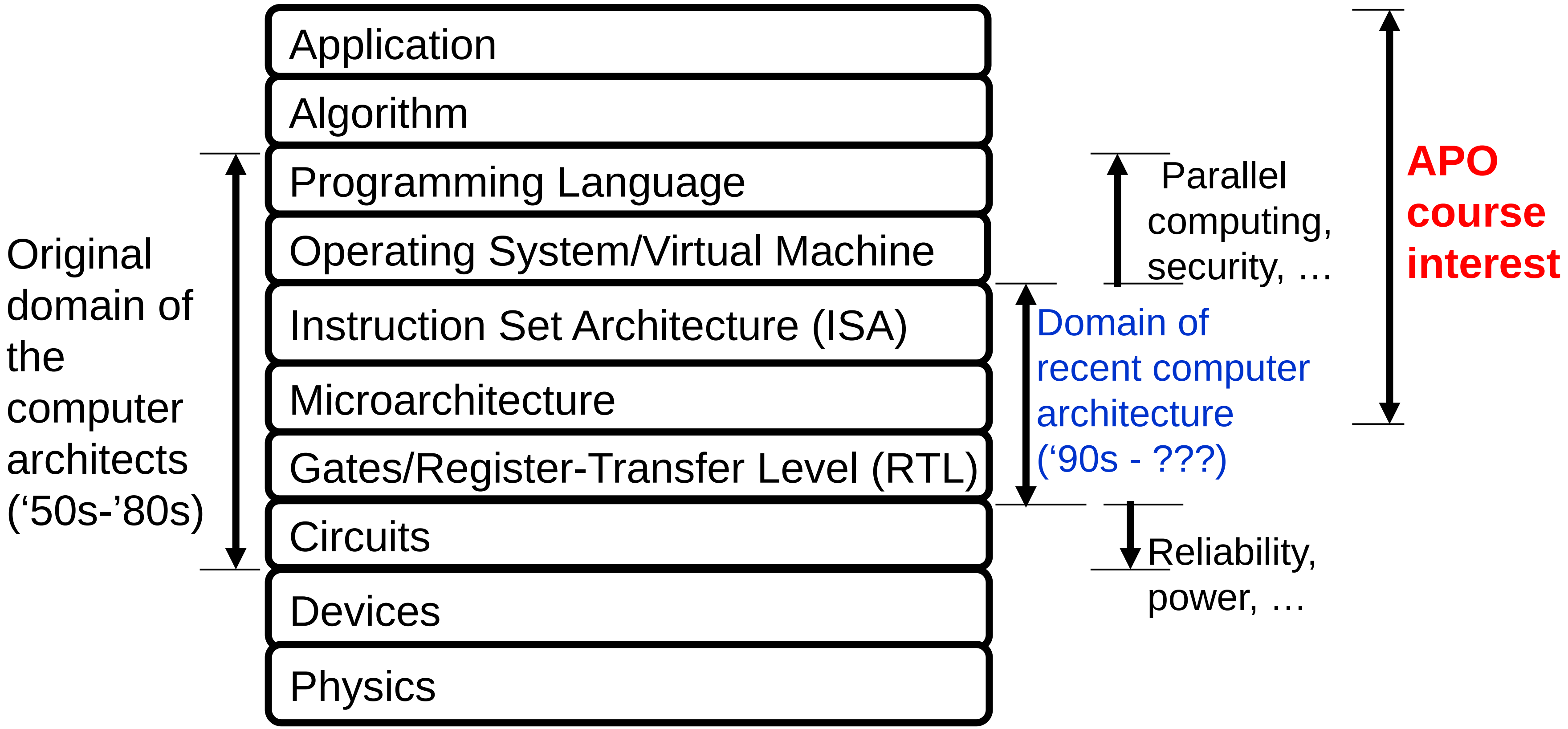
- But in an **embedded** device, it is sometimes necessary to reduce its consumption and cost. There are used very simple processors or microcontrollers, sometimes without real number operations, and programmed with low-level C language.



Applicability of Knowledge and Techniques from the Course

- Applications not only in autonomous control
- In any embedded device - reduce size, consumption, reliability
- In data sciences - considerably reduce runtime and energy savings in calculations
- In the user interface - improving application response
- Practically everywhere…

Computer – Abstraction Levels



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

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.

More Motivation and Examples

- The knowledge is necessary for every programmer who wants to work with medium size data sets or solve little more demanding computational tasks
- No multimedia algorithm can be implemented well without this knowledge
- The 1/3 of the course is focussed even on peripheral access
- Examples
  - Facebook – HipHop for PHP → C++/GCC → machine code
  - BackBerry (RIM) – our consultations for time source
  - RedHat – JAVA JIT for ARM for future servers generation
  - Multimedia and CUDA computations
  - Photoshop, GIMP (data/tiles organization in memory)
  - Knot-DNS (RCU, Copy on write, Cuckoo hashing,)

The Course's Background and Literature

- Course is based on worldwide recognized book and courses; evaluation Graduate Record Examination – GRE

  Paterson, D., Henessy, J.: Computer Organization and Design, The HW/SW Interface. Elsevier, ISBN: 978-0-12-370606-5

  - John L. Henessy – president of Stanford University, one of founders of MIPS Computer Systems Inc.
  - David A. Patterson – leader of Berkeley RISC project and RAID disks research
- Our experience even includes distributed systems, embedded systems design (of mobile phone like complexity), peripherals design, cooperation with carmakers, medical and robotics systems design

Topics of the Lectures

- Architecture, structure and organization of computers and its subsystems.
- Floating point representation
- Central Processing Unit (CPU)
- Memory
- Pipelined instruction execution
- Input/output subsystem of the computer
- Input/output subsystem (part 2)
- External events processing and protection
- Processors and computers networks
- Parameter passing
- Classic register memory-oriented CISC architecture
- INTEL x86 processor family
- CPU concepts development (RISC/CISC) and examples
- Multi-level computer organization, virtual machines

Topics of Seminaries

- 1 - Introduction to the lab
- 2 - Data representation in memory and floating point
- 3 - Processor instruction set and algorithm rewriting
- 4 - Hierarchical concept of memories, cache - part 1
- 5 - Hierarchical concept of memories, cache - part 2
- 6 - Pipeline and gambling
- 7 - Jump prediction, code optimization
- 8 - I / O space mapped to memory and PCI bus
- 9 - HW access from C language on MZ_APO
- Semestral work

Classification and Conditions to Pass the Subject

Conditions for assessment:

Category	Points	Required minimum	Remark
4 homeworks	36	12	3 of 4
Activity	8	0	
Team project	24	5	
Sum	**60 (68)**	30	

Exam:

Category	Points	Required minimum
Written exam part	30	15
Oral exam part	+/- 10	0

Grade	Points range
A	90 and more
B	80 - 89
C	70 - 79
D	60 - 69
E	50 - 59
F	less than 50

The First Lecture Contents

- Number representation in computers
  - numeral systems
  - integer numbers, unsigned and signed
  - boolean values
- Basic arithmetic operations and their implementation
  - addition, subtraction
  - shift right/left
  - multiplication and division

Motivation: What is the output of next code snippet?

```
int main() {
  int a = -200;
  printf("value: %u = %d = %f = %c \n", a, a,
  *((float*)(&a)), a);

  return 0;
}
```

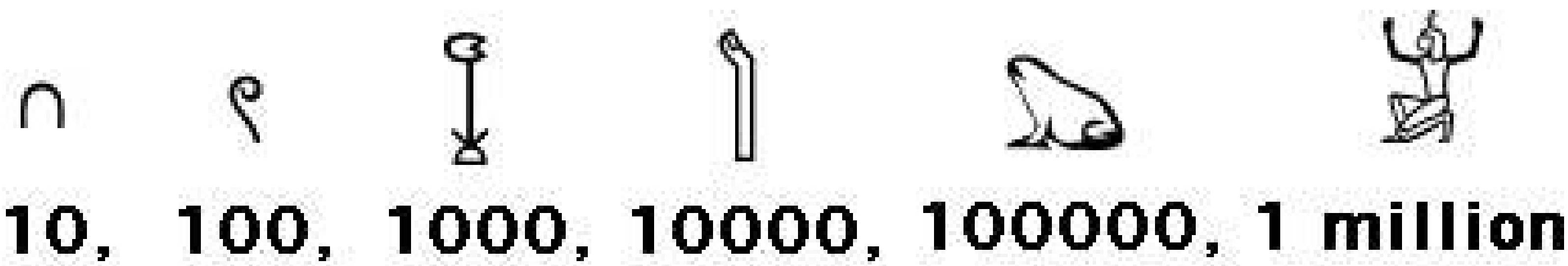
value: 4294967096 = -200 = nan = 8

and memory content is: 0x38 0xff 0xff 0xff

when run on little endian 32 bit CPU.

Non-positional numbers ☺

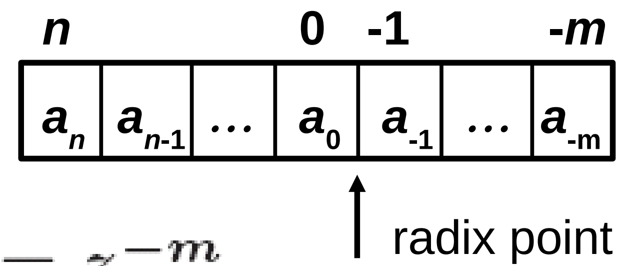
http://diameter.si/sciquest/E1.htm



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 $\langle \mathbf{0,z^{n+m+1}-1} \rangle$
- 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:

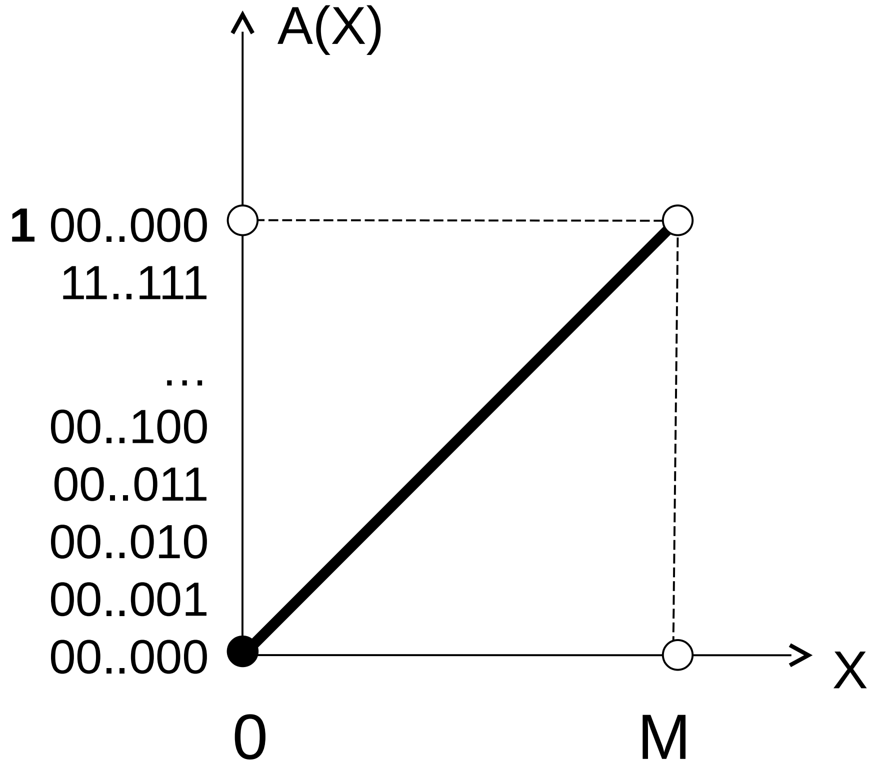
`unsigned int`

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 … $4294967295_d = 0xFFFFFFFF_h$

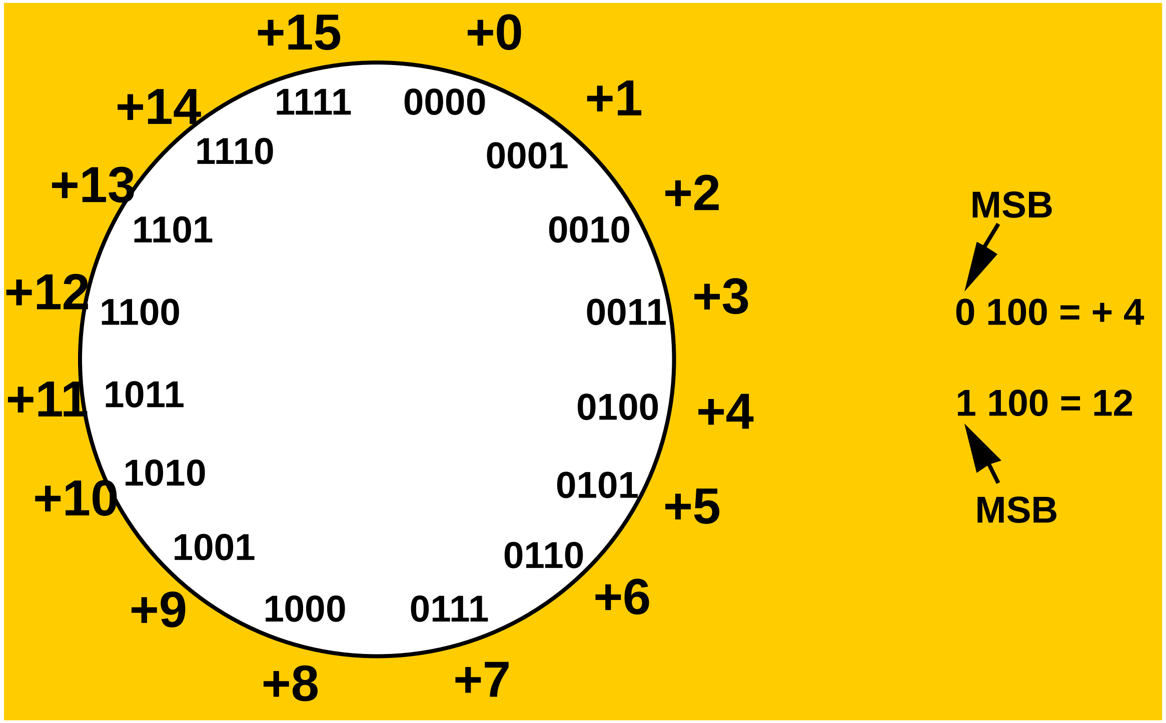
Unsigned Integer



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

[Seungryoul Maeng:Digital Systems]

Signed Numbers

Language C:

`int`

`signed int`

Signed Integer Numbers

- Work with negative numbers is required for many applications
- When appropriate representation is used then same hardware (with minor extension) can be used for many operations with signed and unsigned numbers
- Possible representations
  - sign-magnitude code, direct representation, sign bit
  - two's complement
  - ones' complement
  - excess-K, offset binary or biased representation

Two's Complement (Complement to Module)

- The most frequent code
- The unsigned sum of two opposite numbers representations with the same absolute value is 00000000H!
- Transform to the representation

  D(A) = A   iff A≥0

  D(A) = Z-|A| iff A<0
- The name is misleading, it is module (Z) complement actually
- Negative value can be encoded as digit by digit z-1-a_i and then add 1
- For z=2, bit by bit complement (0→1, 1→0) and addition of 1

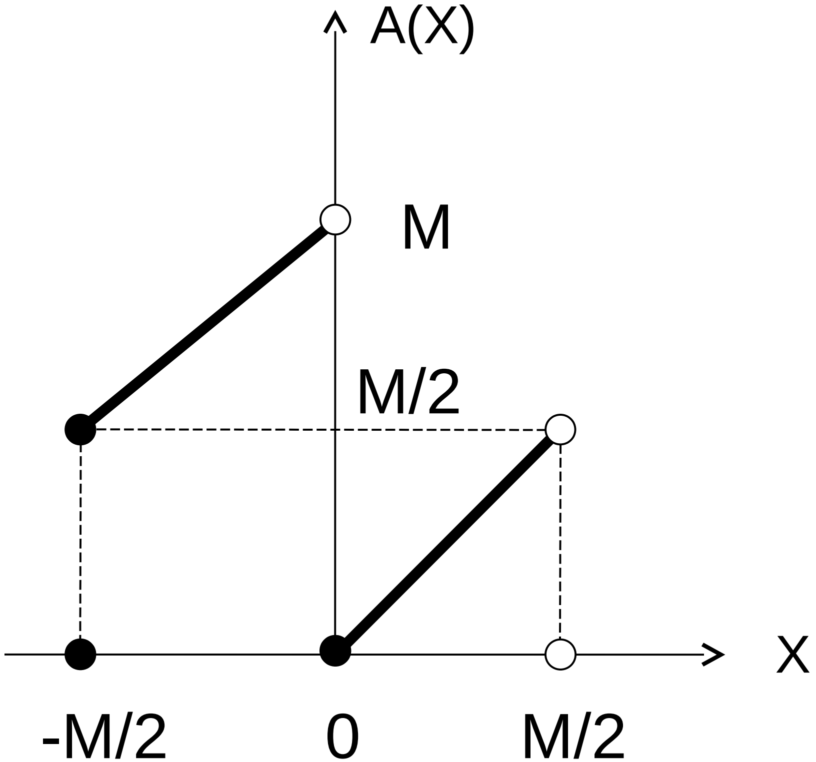
Decimal value	4 bit two's compliment
6	0110
-6	1010

Two's Complement

For **N** bits used for representation is represented value range**:**

<-2^(N-1) , 2^(N-1) -1>

$$\langle -2^{N-1},\ 2^{N-1}-1 \rangle$$

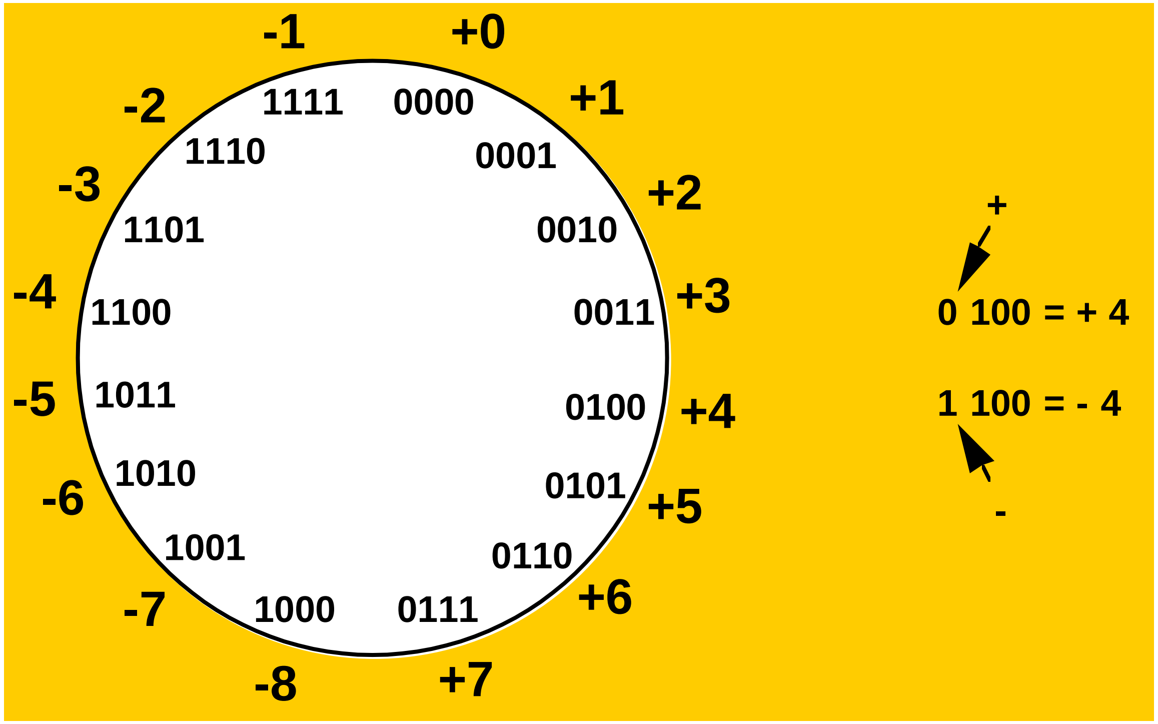


Binární hodnota	Dvojkový doplněk
00000000	$0_{(10)}$
00000001	$1_{(10)}$
⋮	⋮
01111101	$125_{(10)}$
01111110	$126_{(10)}$
01111111	$127_{(10)}$
10000000	$-128_{(10)}$
10000001	$-127_{(10)}$
10000010	$-126_{(10)}$
⋮	⋮
11111101	$-3_{(10)}$
11111110	$-2_{(10)}$
11111111	$-1_{(10)}$

Two's Complement - Examples

- Advantages
  - Continuous range when cyclic arithmetic is considered
  - Single and one to one mapping of value 0
  - Same HW for signed and unsigned adder
- Disadvantages
  - Asymmetric range $(-(-{}^{1}/_{2}Z))$
- Examples:
  - $0_D = 00000000_H$,
  - $1_D = 00000001_H$, $-1_D = \mathrm{FFFFFFFF}_H$,
  - $2_D = 00000002_H$, $-2_D = \mathrm{FFFFFFFE}_H$,
  - $3_D = 00000003_H$, $-3_D = \mathrm{FFFFFFFD}_H$,

Twos Complement (In Czech: Druhý doplněk)



Only one representation for 0

One more negative number than positive number

[Seungryoul Maeng:Digital Systems]

Two's Complement and the C Language

- Two's complement is most used signed integer numbers representation in computers
- Complement arithmetic is often used as its synonym
- “C” programing language speaks about integer numeric type without sign as *unsigned integers* and they are declared in source code as `unsigned int`.
- The numeric type with sign is simply called *integers* and is declared as `signed int`.
- Considerations about overflow and underflow are described later.

Two's Complement and the C Language

- The typical ranges for C language integers on 32-bit targets:

Bytes typical	Bits min/typ	Type in C	Range	
			Min	Max
1	8/8	unsigned char	0	255
1	8/8	char	-128	127
2	16/16	unsigned short int	0	65,535
2	16/16	short int	−32,768	32,767
4	16/32	Unsigned int	0	4,294,967,295
4	16/32	int	−2,147,483,648	2,147,483,647
4	32/32	unsigned long int	0	4,294,967,295
4	32/32	long int	−2,147,483,648	2,147,483,647
8	64/64	unsigned long long int	0	18,446,744,073,709,551,615
8	64/64	long long int	−9,223,372,036,854,775,808	+9,223,372,036,854,775,807

Two's Complement – Addition and Subtraction

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

Other Possibilities

Integer – Biased Representation

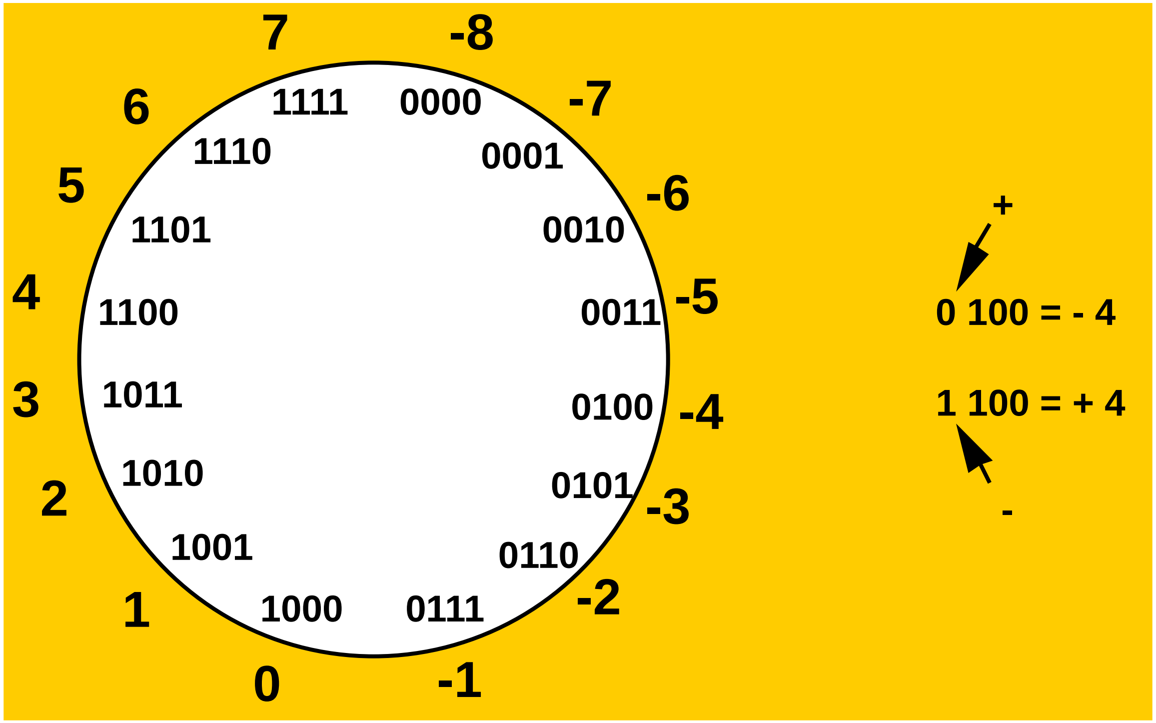
- Known as excess-K or offset binary as well
- Transform to the representation

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

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

[Seungryoul Maeng:Digital Systems]

Addition and Subtraction for the Biased Representation

- Short note about other signed number representation

$$\mathcal{A}(A+B) = \mathcal{A}(A) + \mathcal{A}(B) - K$$
$$\mathcal{A}(A-B) = \mathcal{A}(A) - \mathcal{A}(B) + K$$

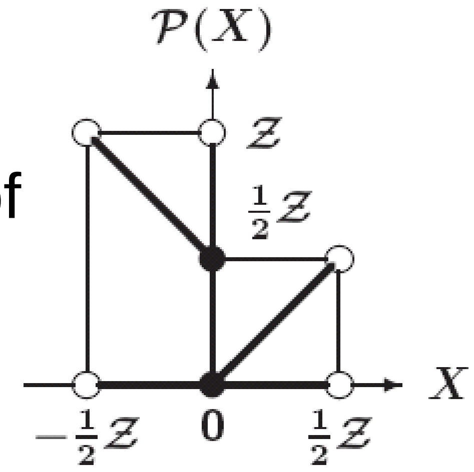
- Overflow detection
  - for addition:

    same sign of addends and different result sign
  - for subtraction:

    signs of minuend and subtrahend are opposite and sign of the result is opposite to the sign of minuend

Integer – Sign-Magnitude Code

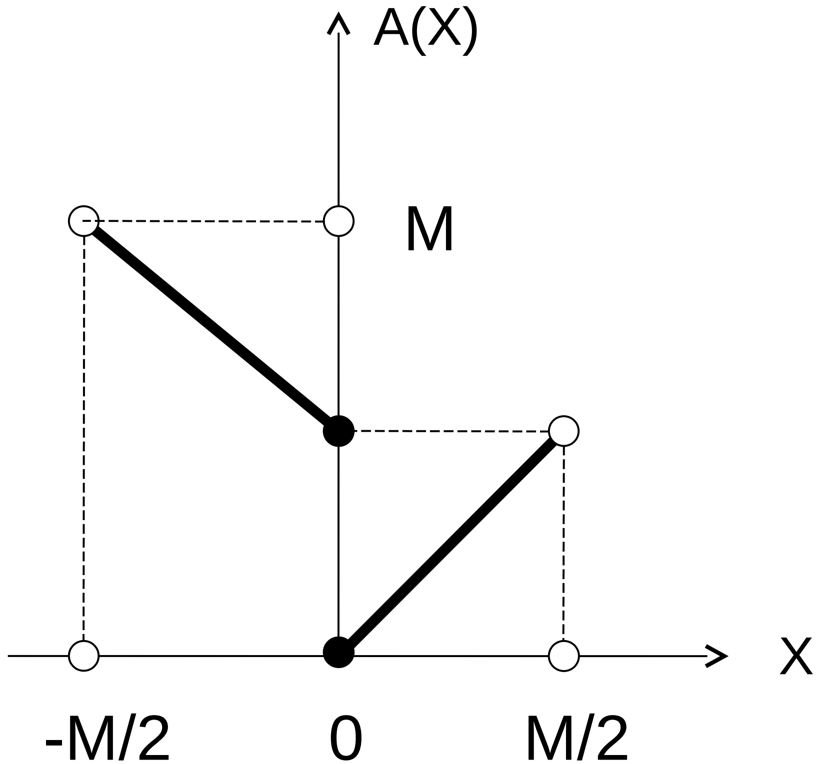
- Sign and magnitude of the value (absolute value)
- Natural to humans -1234, 1234
- One (usually most significant – MSB) bit of the memory location is used to represent the sign
- Bit has to be mapped to meaning
- Common use 0 ≈ "+", 1 ≈ "-"
- Disadvantages:
  - When location is **k** bits long then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



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

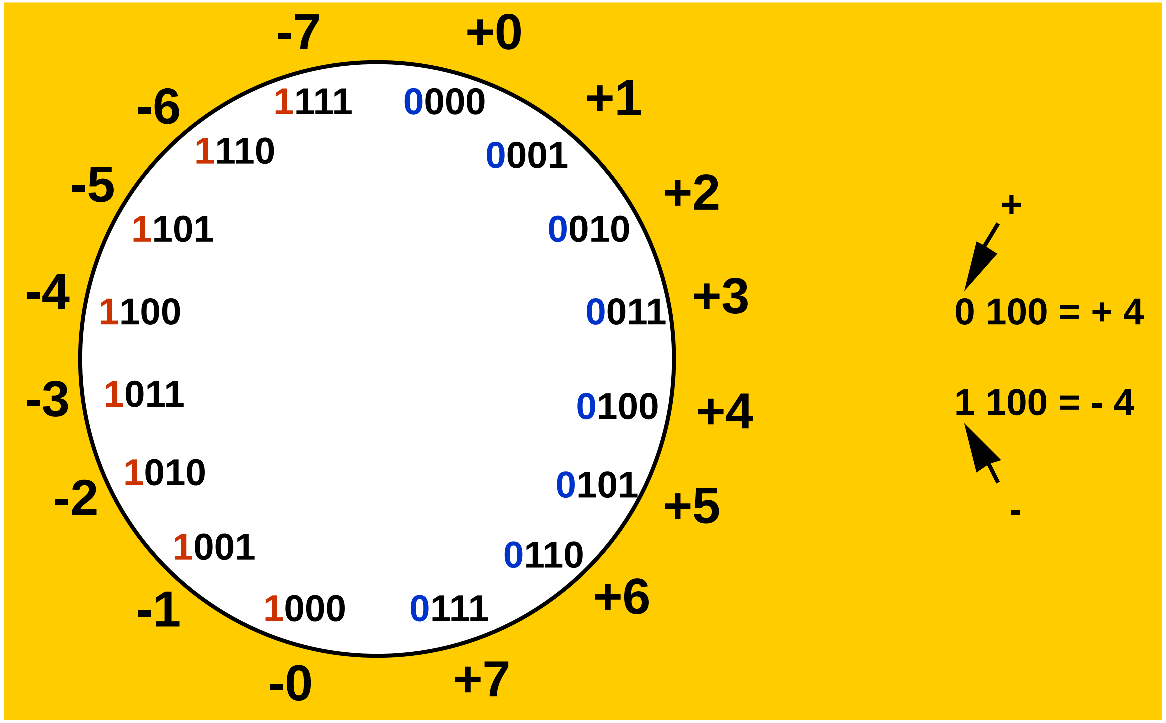
Sign and Magnitude Representation

<-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*

[Seungryoul Maeng:Digital Systems]

Integers – Ones' Complement

- Transform to the representation

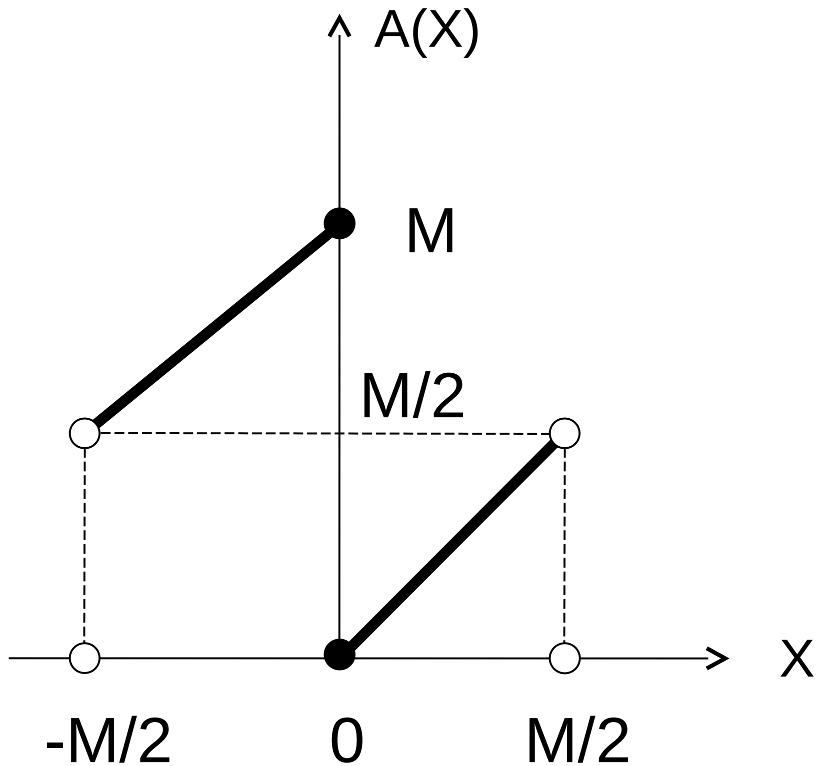
  $-2^{n-1}+1 \ldots 0 \ldots 2^{n-1}-1$

  D(A) = A iff $A \geq 0$

  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

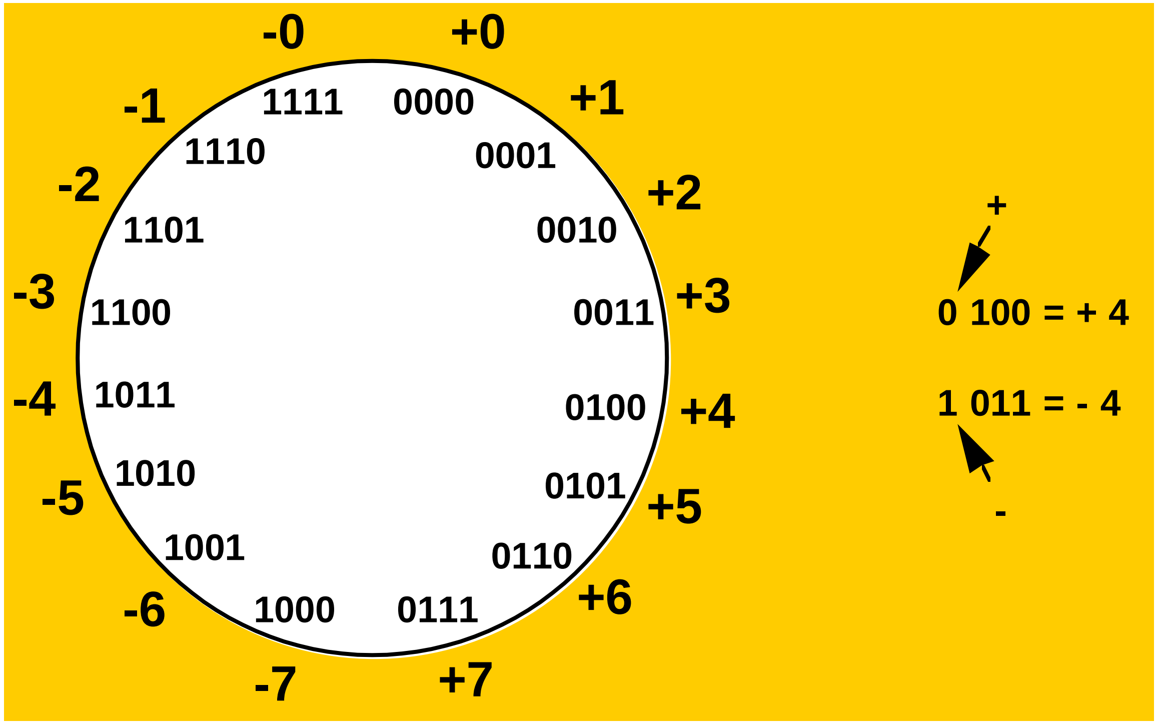
Ones Complement

<-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)

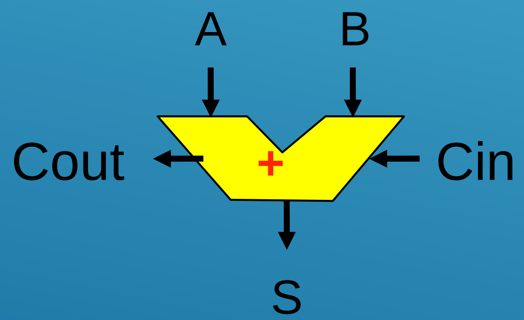


Still two representations of 0! This causes some problems

Some complexities in addition, nowadays nearly not used

[Seungryoul Maeng:Digital Systems]

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



OPERATION WITH INTEGERS

Direct Realization of Adder as Logical Function

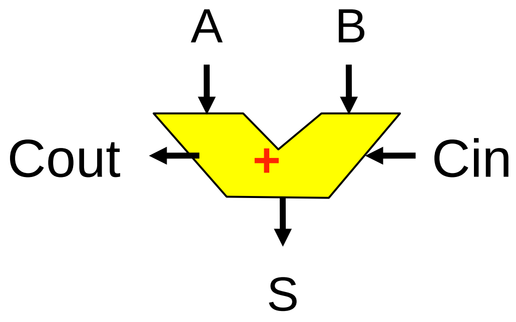
	Number of logic operations
bit width	for calculating sum
1	3
2	22
3	89
4	272
5	727
6	1567
7	3287
8	7127
9	17623
10	53465
11	115933

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

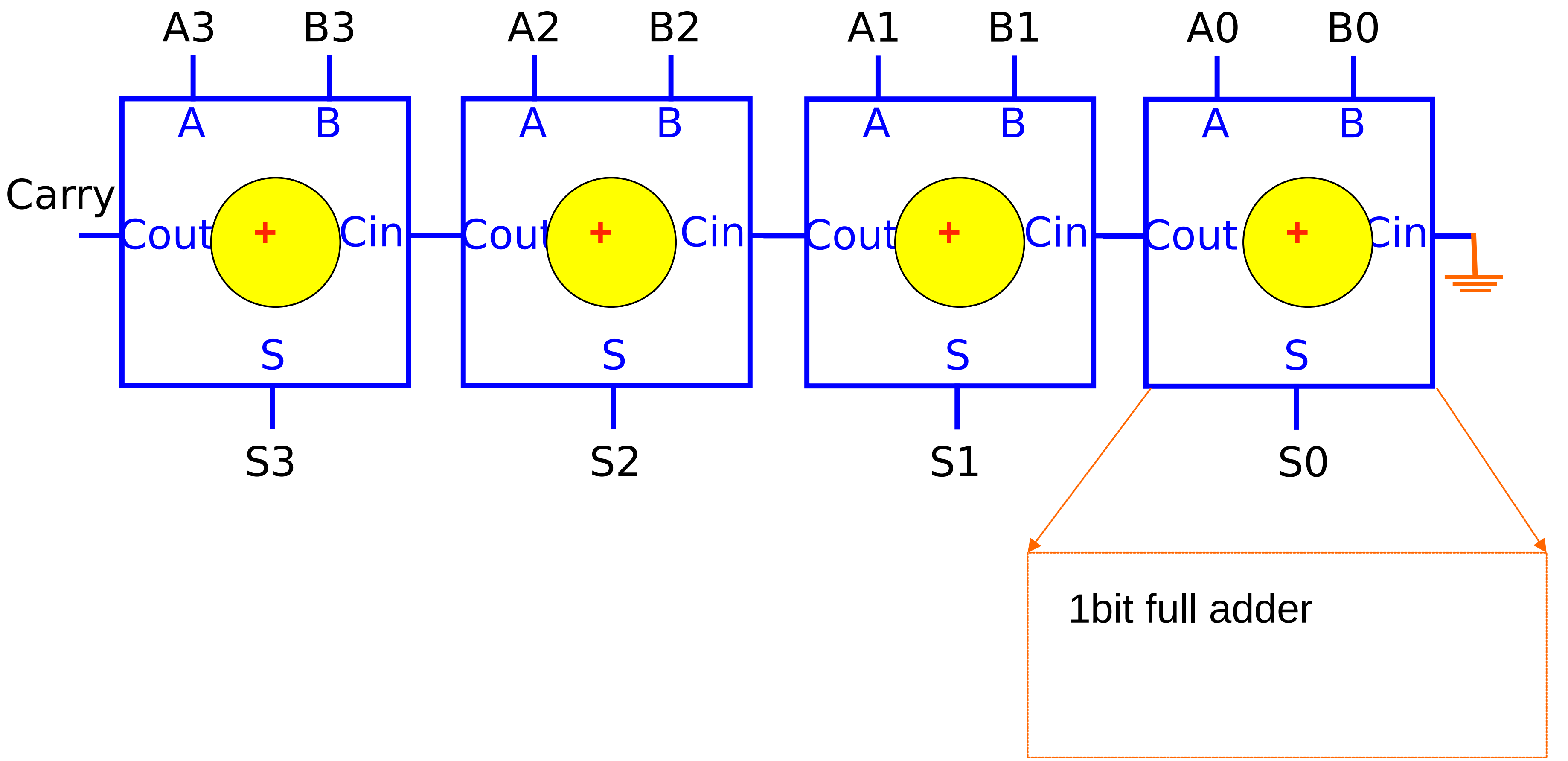
The calculation was performed by BOOM logic minimizer created at the Department of Computer Science CTU-FEE

1bit Full Adder

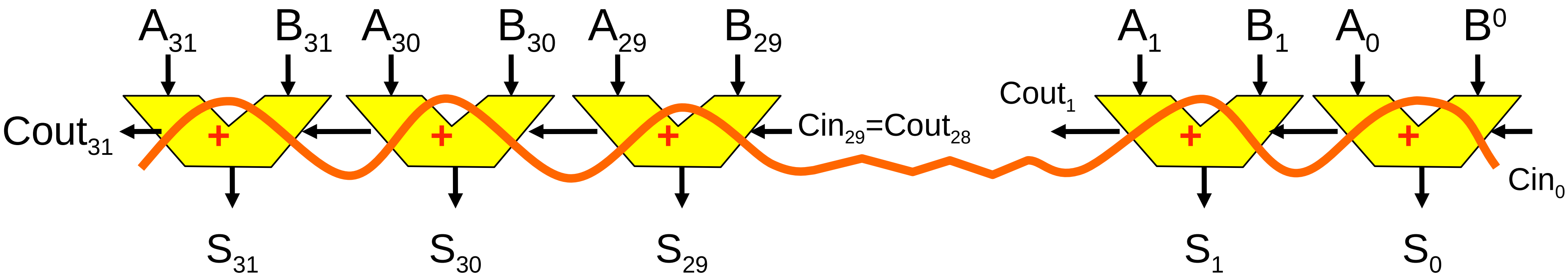
A	0	0	1	1	0	0	1	1
+B	0	1	0	1	0	1	0	1
Sum	00	01	01	10	00	01	01	10
+ Carry-In	0	0	0	0	1	1	1	1
CarryOut Sum	00	01	01	10	01	10	10	11



Simple Adder



Simple Adder



Simplest N-bit adder

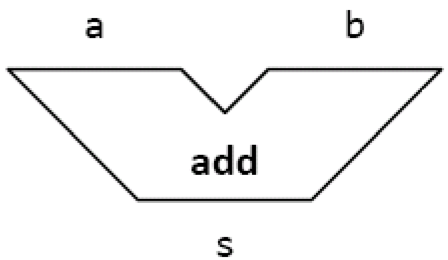
we chain 1-bit full adders

"Carry" ripple through their chain

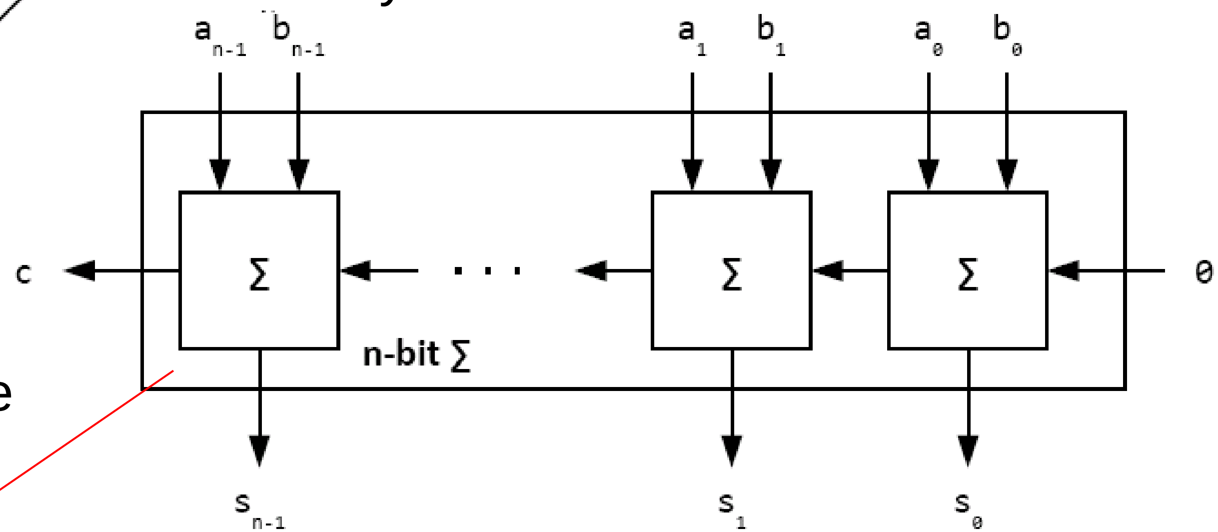
Minimal number of logical elements

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

Hardware of Ripple-Carry Adder

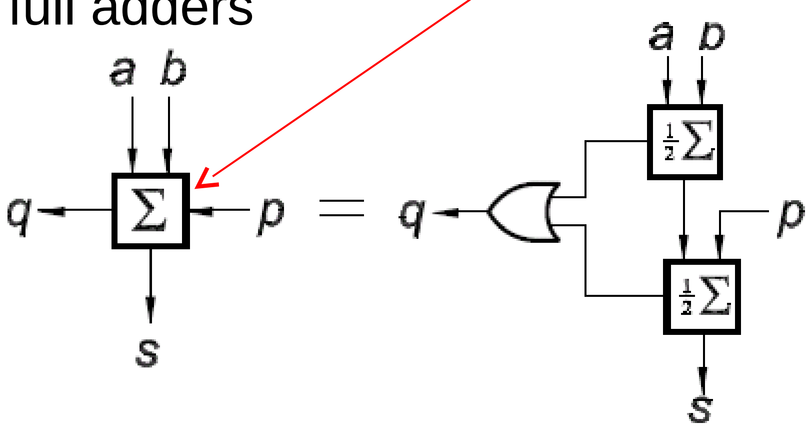


Common symbol for adder

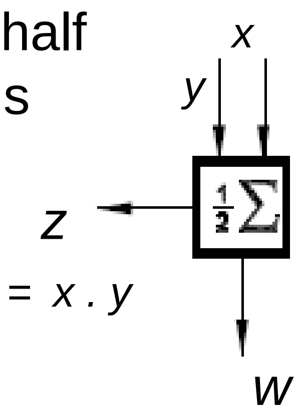


Internal structure

Realized by 1-bit full adders



where half adder is



$z = x \,.\, y$

$w = x \oplus y$

Fast Parallel Adder Realization and Limits

- The previous, cascade based adder is slow – carry propagation delay
- The parallel adder is combinatorial circuit, it can be realized through sum of minterms (product of sums), two levels of gates (wide number of inputs required)
- But for 64-bit adder 10^{20} gates is required

Solution #1

- Use of carry-lookahead circuits in adder combined with adders without carry bit

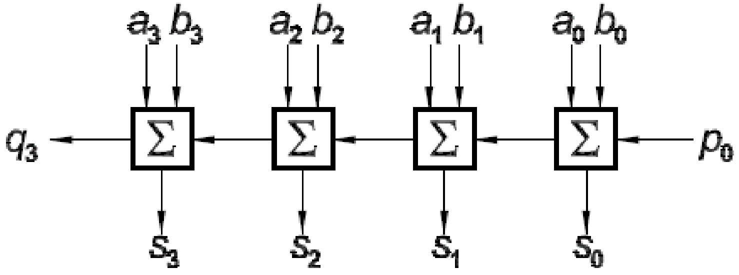
Solution #2

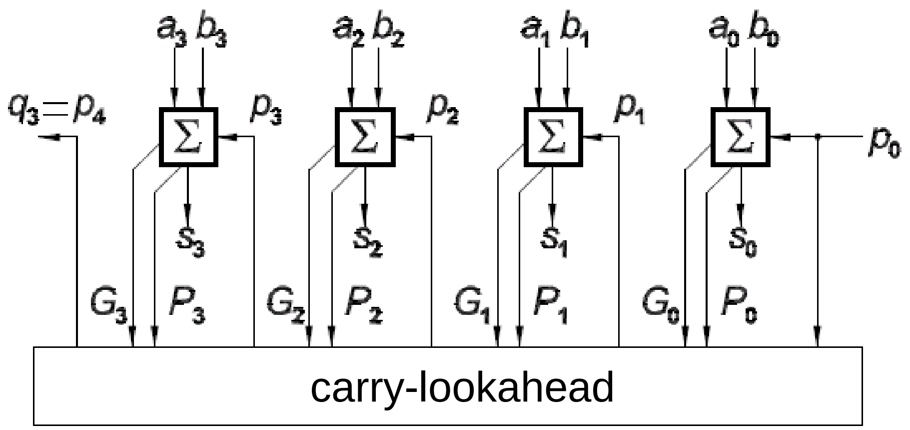
- Cascade of adders with fraction of the required width

Combination (hierarchy) of #1 and #2 can be used for wider inputs – tree of CLAs to speed up computation to O(log(n))

Speed of the Adder

- Parallel adder is combinational logic/circuit. Is there any reason to speak about its speed? Try to describe!
- Yes, and it is really slow. Why?
- Possible enhancement – adder with carry-lookahead (CLA) logic!





CLA – Carry-lookahead

- Adder combined with CLA provides enough speedup when compared with parallel ripple-carry adder and yet number of additional gates is acceptable
- CLA for 64-bit adder increases hardware price for about 50% but the speed is increased (signal propagation time decreased) 9 times.
- The result is significant speed/price ratio enhancement.

The Basic Equations for the CLA Logic

- Let:
  - the generation of carry on position (bit) j is defined as:

$$g_j = x_j y_j$$

  - the need for carry propagation from previous bit:

$$p_j = x_j \oplus y_j = x_j \bar{y}_j \vee \bar{x}_j y_j$$

- Then:
  - the result of sum for bit j is given by:

$$s_j = c_j \left(\overline{x_j \oplus y_j}\right) \vee \bar{c}_j \left(x_j \oplus y_j\right) = c_j \bar{p}_j \vee \bar{c}_j p_j = p_j \oplus c_j$$

  - and carry to the higher order bit (j+1) is given by:

$$c_{j+1} = x_j y_j \vee \left(x_j \oplus y_j\right) c_j = g_j \vee p_j c_j$$

CLA

The carry can be computed as:

$c_1 = g_0 \vee p_0c_0$

$c_2 = g_1 \vee p_1c_1 = g_1 \vee p_1(g_0 \vee p_0c_0) = g_1 \vee p_1g_0 \vee p_1p_0c_0$

$c_3 = g_2 \vee p_2c_2 = g_2 \vee p_2(g_1 \vee p_1g_0 \vee p_1p_0c_0) = g_2 \vee p_2g_1 \vee p_2p_1g_0 \vee p_2p_1p_0c_0$

$c_4 = g_3 \vee p_3c_3 = \ldots = g_3 \vee p_3g_2 \vee p_3p_2g_1 \vee p_3p_2p_1g_0 \vee p_3p_2p_1p_0c_0$

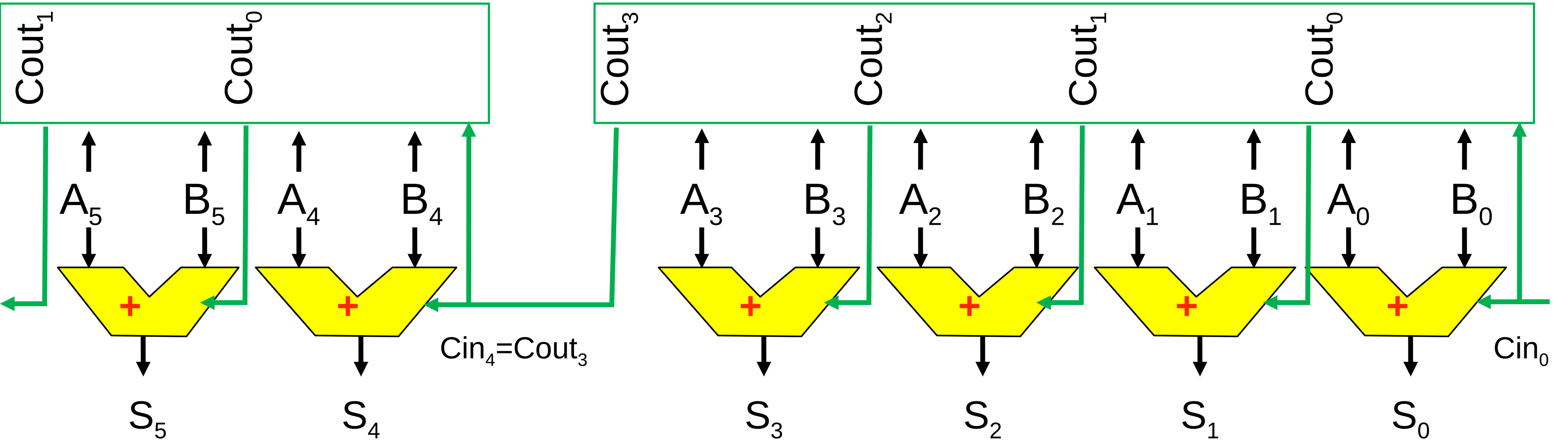
$c_5 = \ldots$

Description of the equation for c_3 as an example:

The carry input for bit 3 is active **when** carry is generated in bit 2 **or** carry propagates condition holds for bit 2 and carry is generated in the bit 1 **or** both bits 2 and 1 propagate carry and carry is generated in bit 0

32bit CLA "carry look-ahead" adder

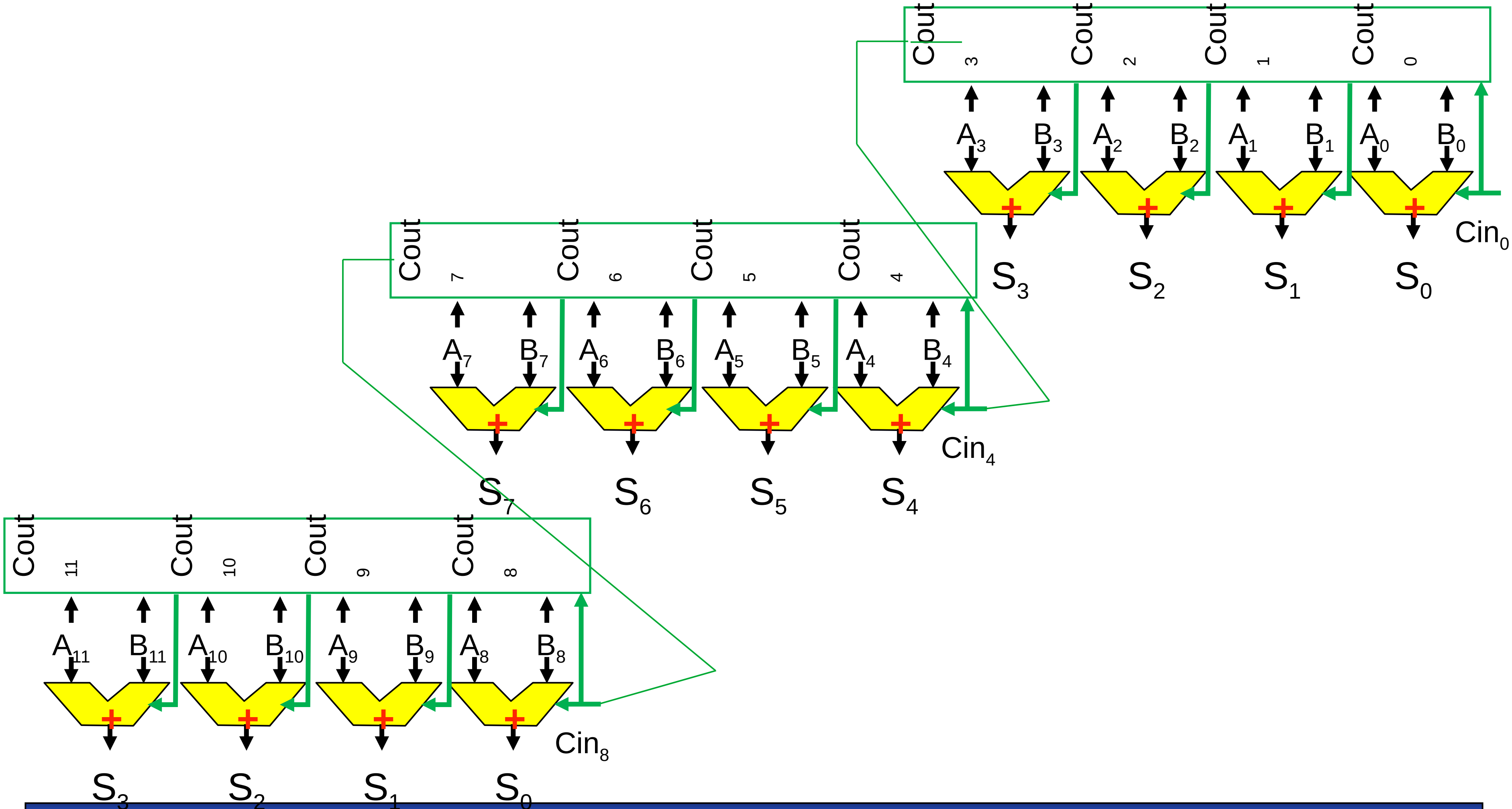
The carry-lookahead adder calculates one or more carry bits before the sum, which reduces the wait time to calculate the result of the larger value bits



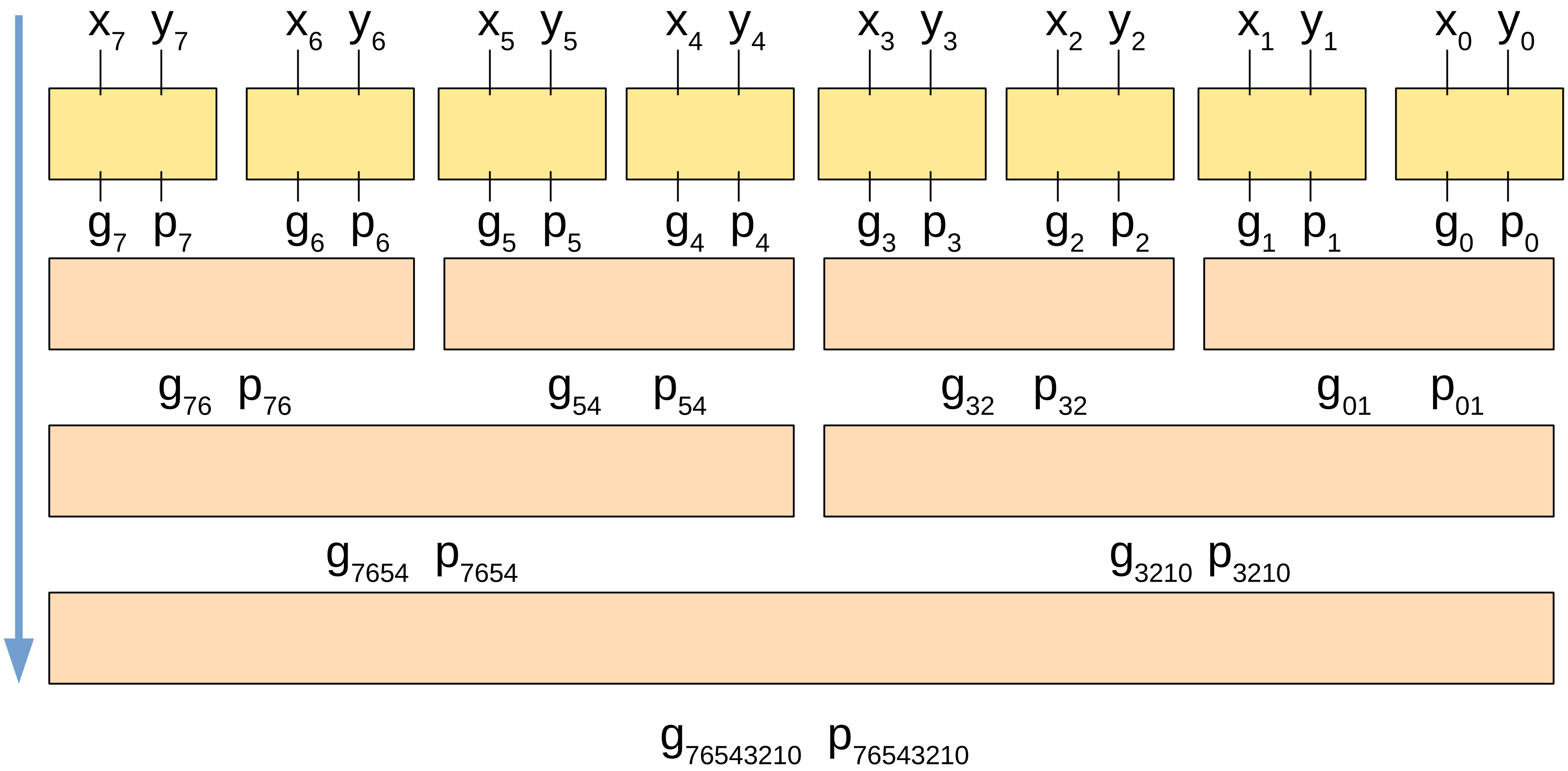
Static "carry look ahead (CLA)" unit for 4 bits

32bit CLA "carry look-ahead" adder

In constant time compute 4-bits sum + carry but still O(n) time



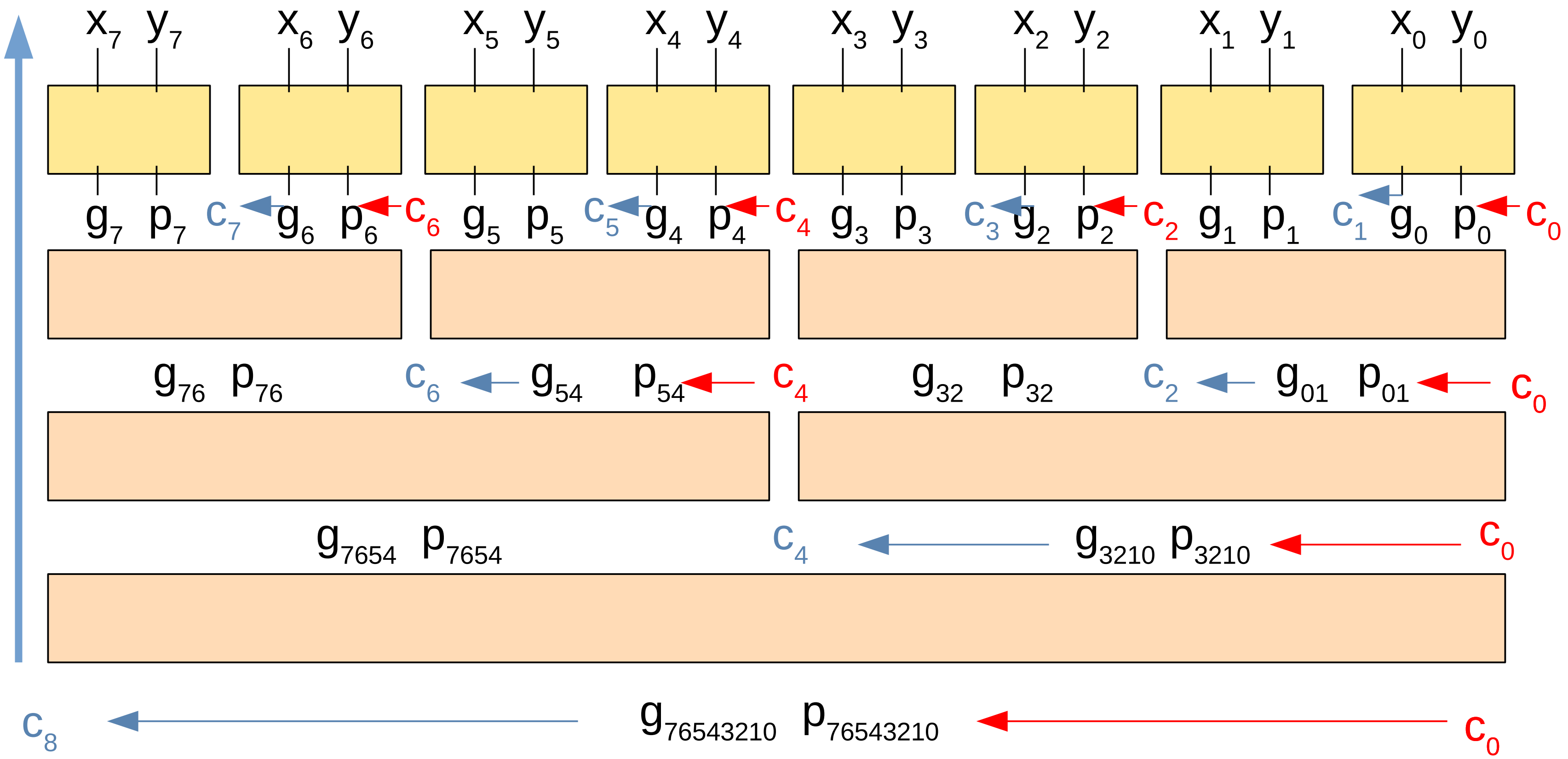
Tree of CLA



$p_{out} = p_{hi}$ and p_{lo}

$g_{out} = g_{hi}$ or (g_{lo} and p_{hi})

Tree of CLA



$c_{out} = g$ or (p and c_{in})

speed up computation to O(log(n))

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

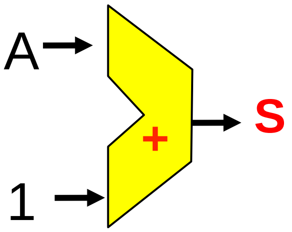
Dec.	Binary 8 4 2 1	+1	Binary 8 4 2 1	-1
0	0000	0001	0000	1111
1	0001	0010	0001	0000
2	0010	0011	0010	0001
3	0011	0100	0011	0010
4	0100	0101	0100	0011
5	0101	0110	0101	0100
6	0110	0111	0110	0101
7	0111	1000	0111	0110
8	1000	1001	1000	0111
9	1001	1010	1001	1000
10	1010	1011	1010	1001
11	1011	1100	1011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

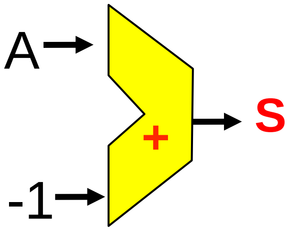


Eq: $S_i = A_i$ xor (A_{i-1} and A_{i-2} and … A_1 and A_0); i=0..n-1

S0=not A0

S1=A1 xor (not A0)

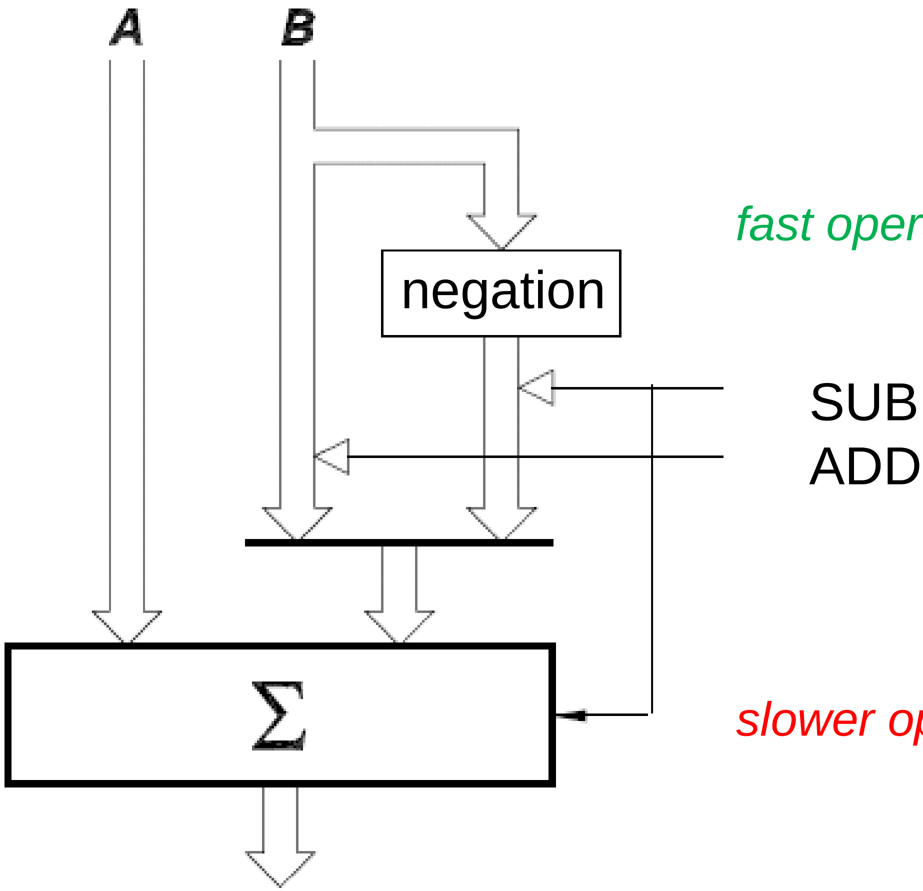
S2=A2 xor (not A1 and not A0)



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



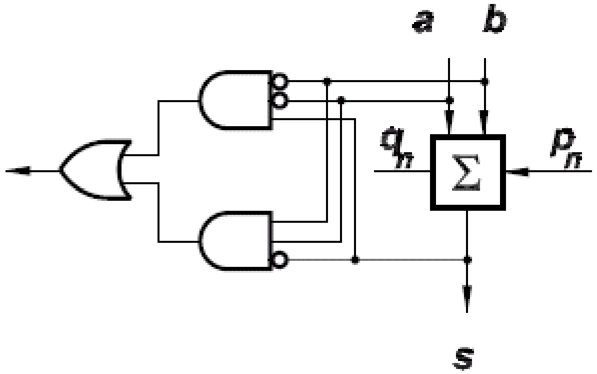
fast operation

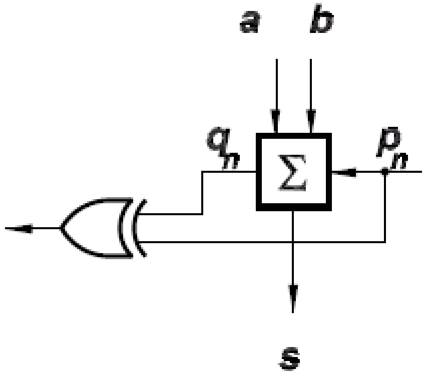
slower operation

Source: X36JPO, A. Pluháček

Arithmetic Overflow (Underflow)

- Result of the arithmetic operation is incorrect because, it does not fit into selected number of the representation bits (width)
- But for the signed arithmetic, it is not equivalent to the carry from the most significant bit.
- The arithmetic overflow is signaled if result sign is different from operand signs if both operands have same sign
- or can be detected with exclusive-OR of carry to and from the most significant bit

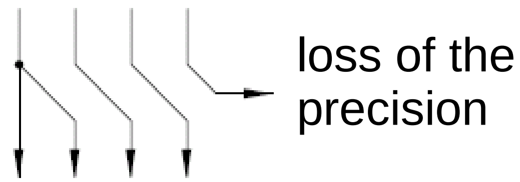


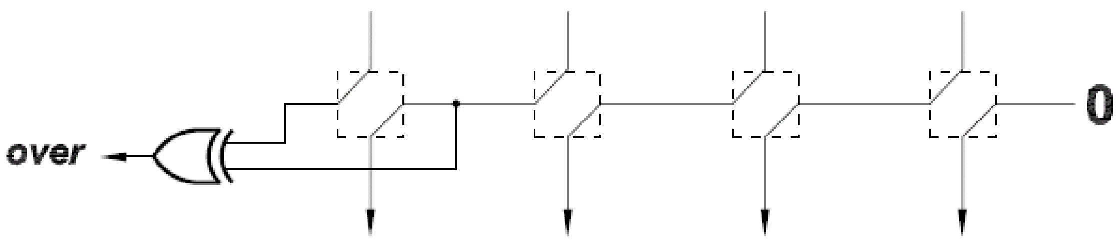


Arithmetic Shift to the Left and to the Right

- arithmetic shift by one to the left/right is equivalent to signed multiply/divide by 2 (digits movement in positional (place-value) representation)

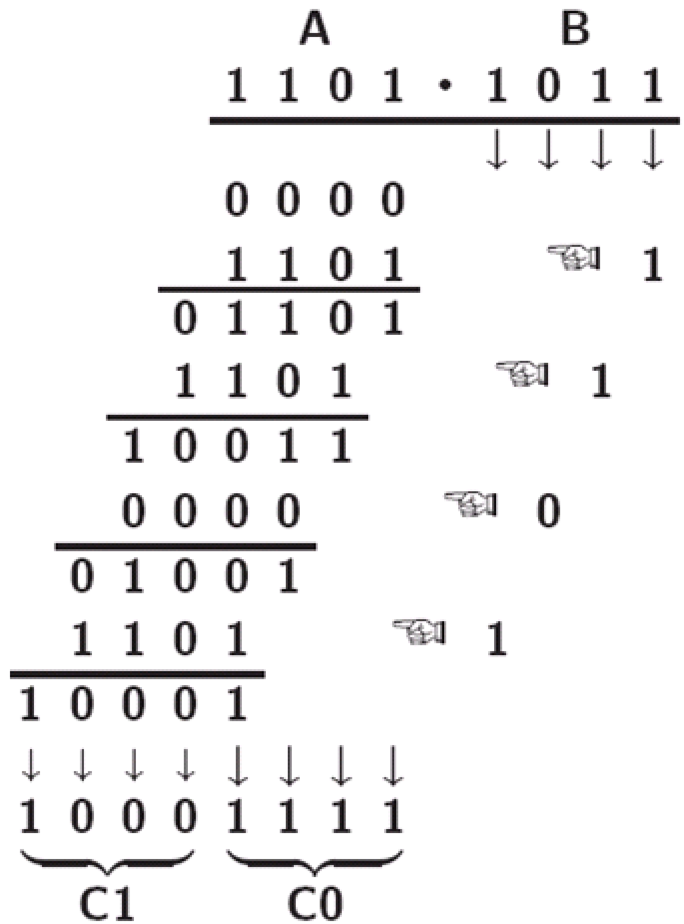
- Notice difference between arithmetic, logic and cyclic shift operations



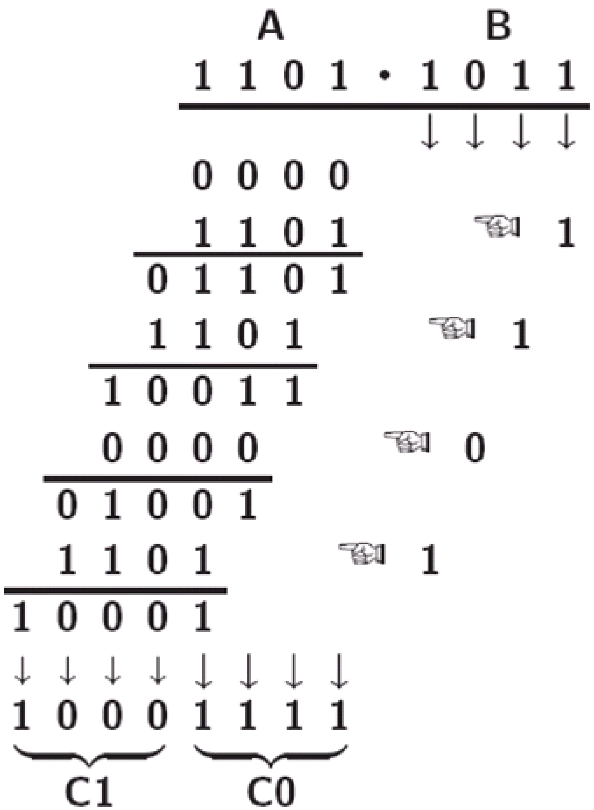


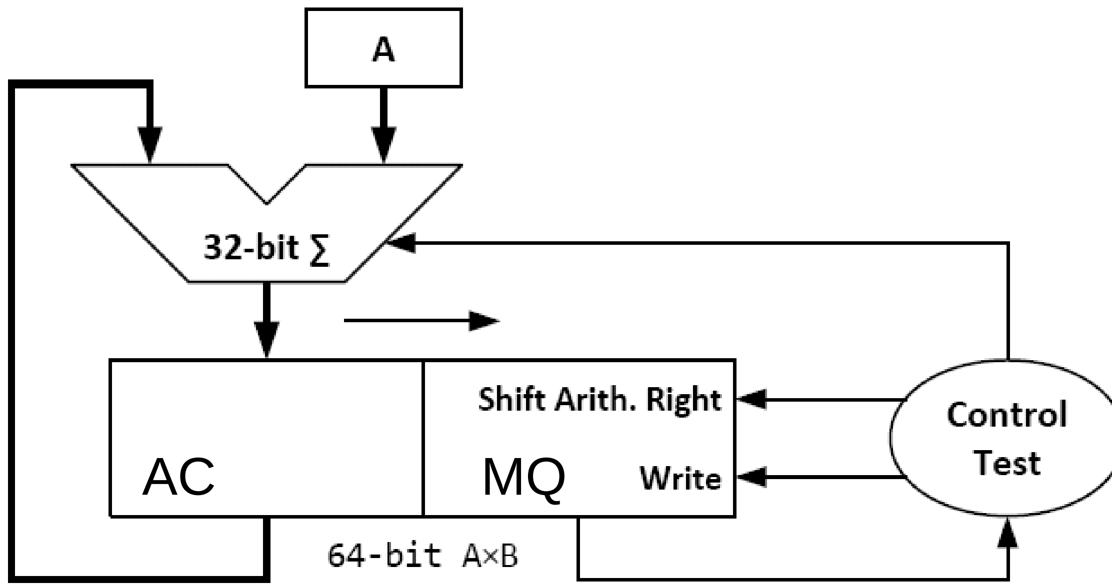
- Remark: Barrel shifter can be used for fast variable shifts

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(MQ0 = = 1)  AC = AC + A;   //  MQ0 = 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. x → A, y → MQ

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

The whole operation: x × y = 110×101 = 011110, (6 × 5 = 30)
Carry,AC can be mapped to AC,MQ.MSB and A masked by MQ.LSB, then if, add and shift steps are combined to the one.

Signed Multiplication by Unsigned HW for two's Complement

One possible solution (for illustration only)

$C = A \cdot B$

Let A and B representations are n bits and result is 2n bits

$D(C) = D(A) \cdot D(B)$
– (D(B)<<n) if $A < 0$
– (D(A)<<n) if $B < 0$

Consider for negative numbers

$(2^n+A) \cdot (2^n+B) = 2^{2n}+2^n A + 2^n B + A \cdot B$

where 2^{2n} is out of the result representation, next two elements have to be eliminated if input is negative

Today more often used separate sign evaluation and multiplication of absolute values.

Wallace Tree Based Multiplier – Required Function

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

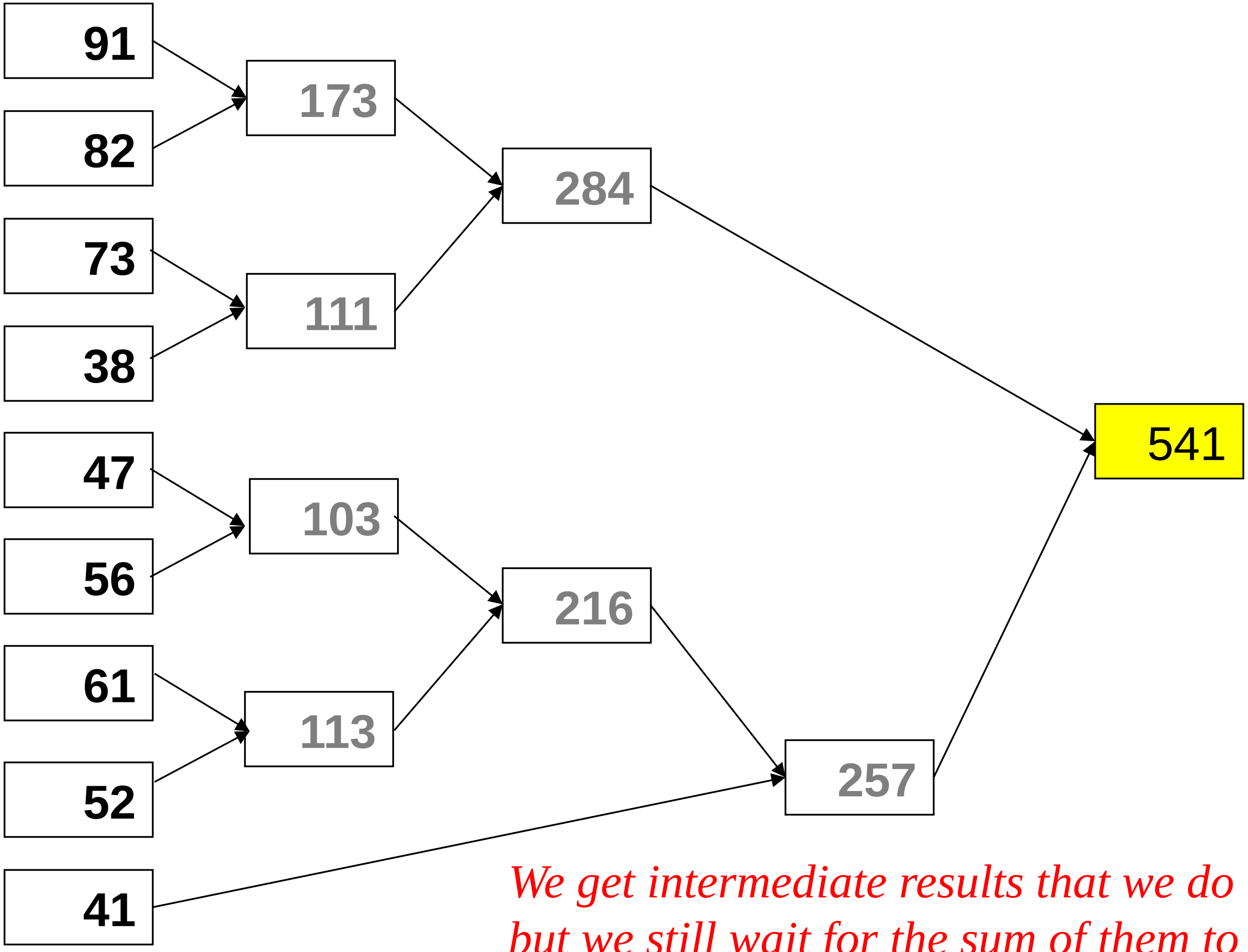
$(x_7\ x_6\ x_5\ x_4\ x_3\ x_2\ x_1\ x_0).(y_7\ y_6\ y_5\ y_4\ y_3\ y_2\ y_1\ y_0) =$

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.

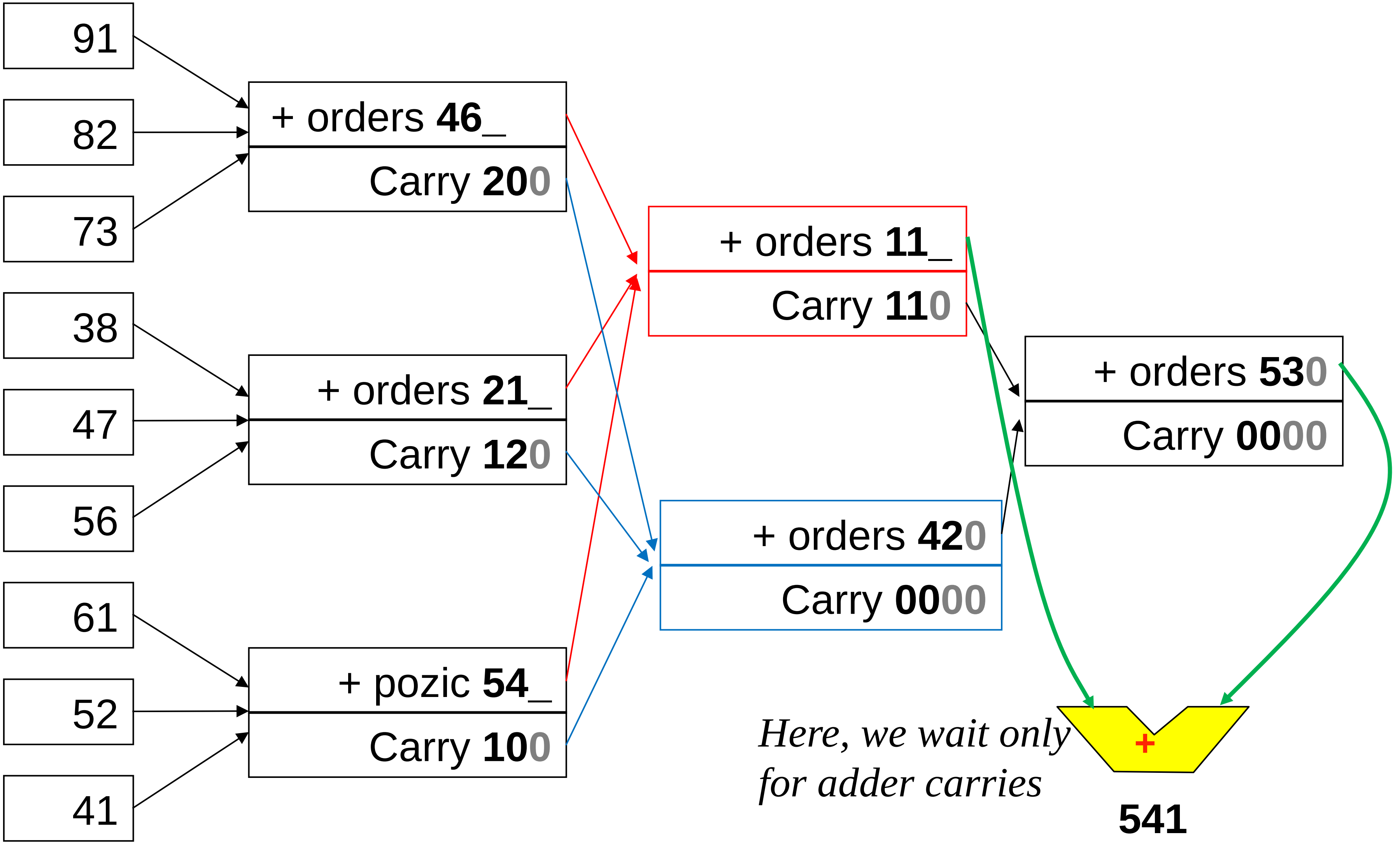
Q = X .Y = P0 + P1 + ... + P7

Idea to Consider – Parallel Adder of 9 Decimal Numbers



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

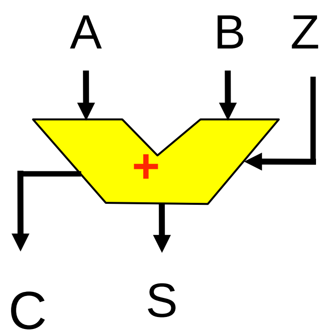
Decadic Carry-save Adder

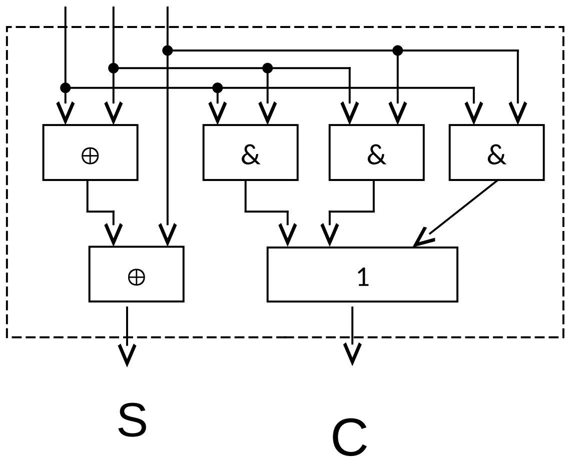


Here, we wait only for adder carries

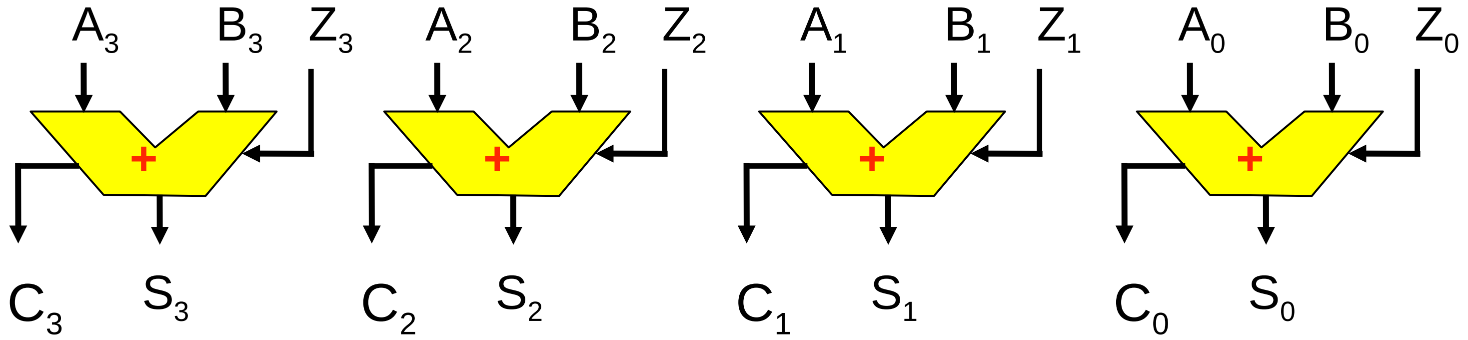
1bit Carry Save Adder

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





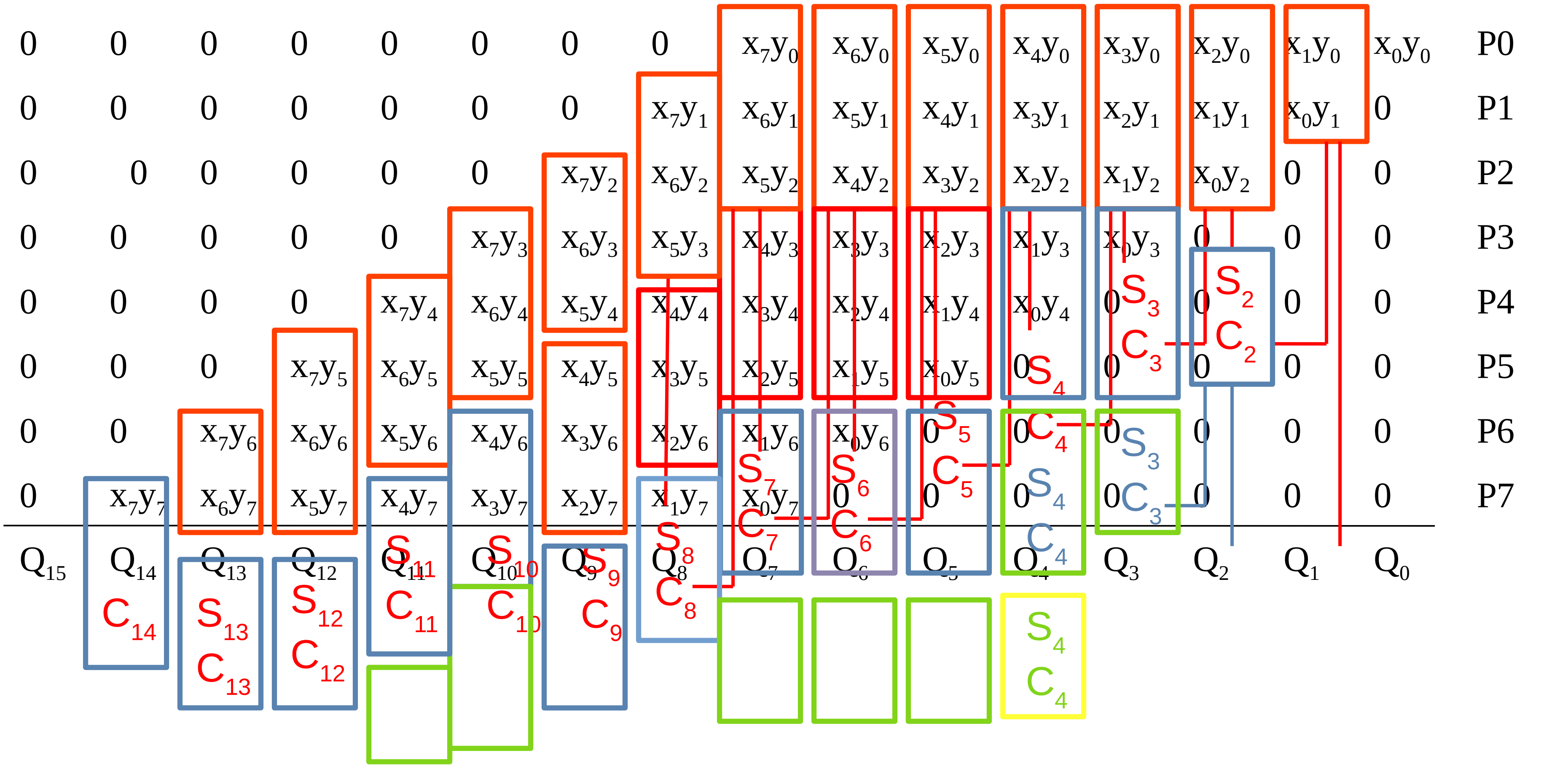
3bit Carry Save Adder



Wallace Tree Based Multiplier – CSA

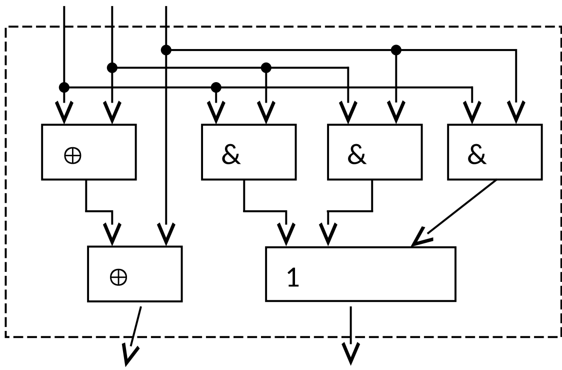
Q=X .Y, X and Y are considered as and 8bit unsigned numbers

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



Wallace Tree Based Fast Multiplier

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



$$S = S^b + C$$

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

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