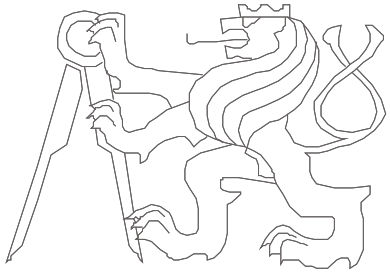
Computer Architectures

Real Numbers and Computer Memory

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.

APO at Dona Lake
84° 28' 45" E, 28° 29' 52" N, 4038m, 2019-11-28
APO at InstallFest (https://installfest.cz/)
2021-03-06 via BigBlueButton running at
50°4'36.682"N, 14°25'4.116"E
QtMIPS Hands on Session to Understand Computer Architectures and Discuss Its Teaching Embedded Linux, FPGA and Motion Control Hands-On

Speed of Arithmetic Operations

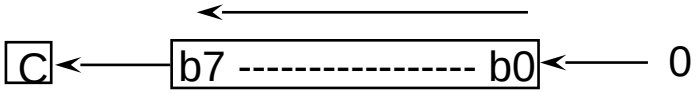
Operation	**C language operator**
Bitwise complement (negation)	`~x`
Multiply and divide by 2^n	`x<<n , x>>n`
Increment, decrement	`++x, x++, --x, x--`
Negate ← complement + increment	`-x`
Addition	`x+y`
Subtraction <- negation + addition	`x-y`
Multiply on hardware multiplier	`x*y`
Multiply on sequential multiplier/SW	
Division	`x/y`

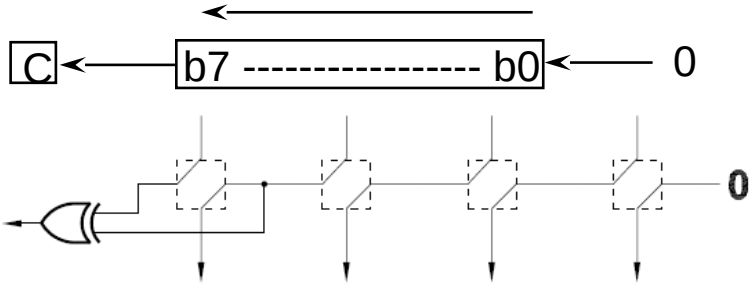
Multiply/Divide by 2

Logical Shift

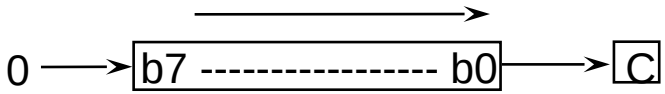
Arithmetic Shift

Multiply by 2

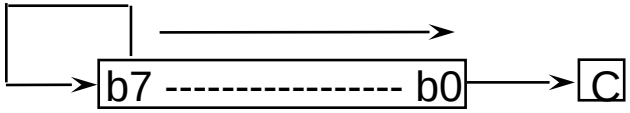


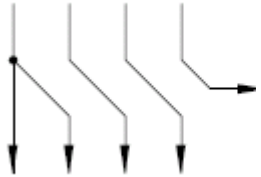


Divide by 2 unsigned



Divide by 2 signed

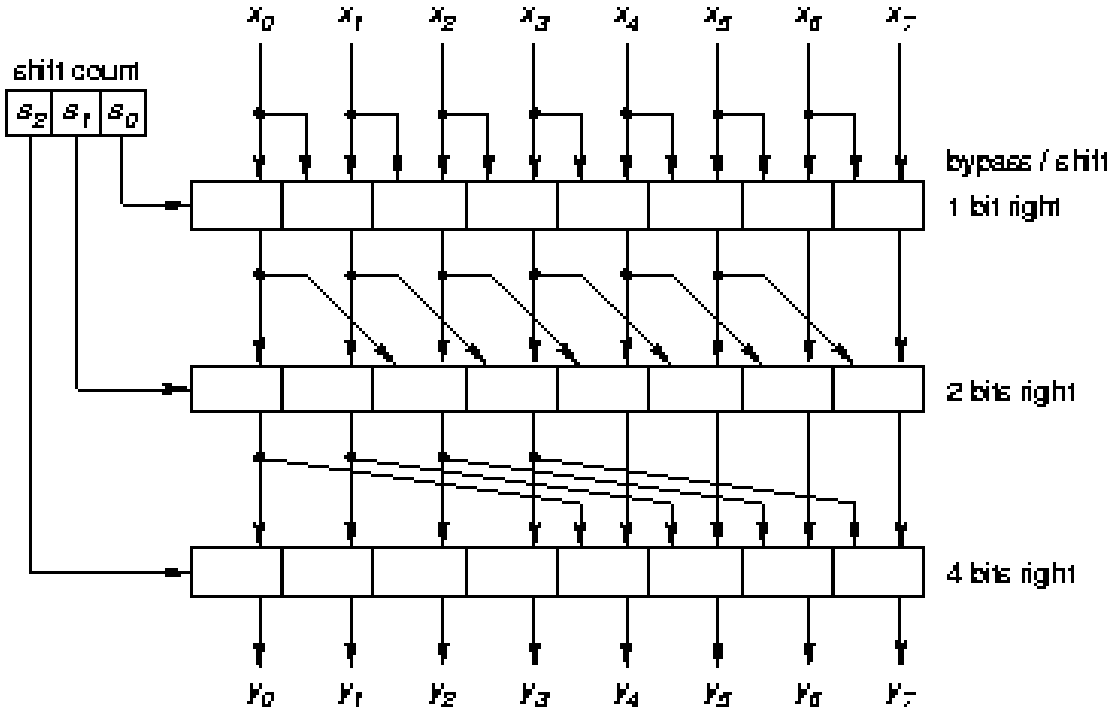




loss of the precision

C represents Carry Flag, it is present only on some processors: x86/ARM yes, MIPS no

Barrel Shifter

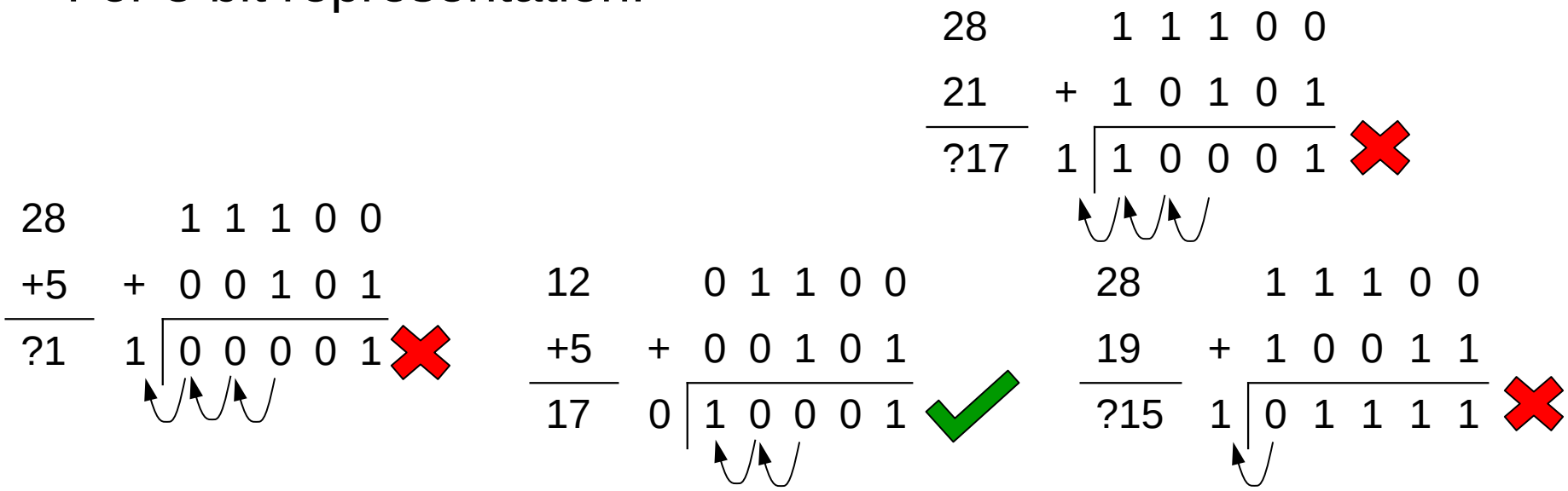


Barrel shifter can be used for fast variable shifts

Overflow of Unsigned Number Binary Representation

- The carry from MSB (the most significant bit) is observed in this case
- The arithmetic result is incorrect because it is out of range.

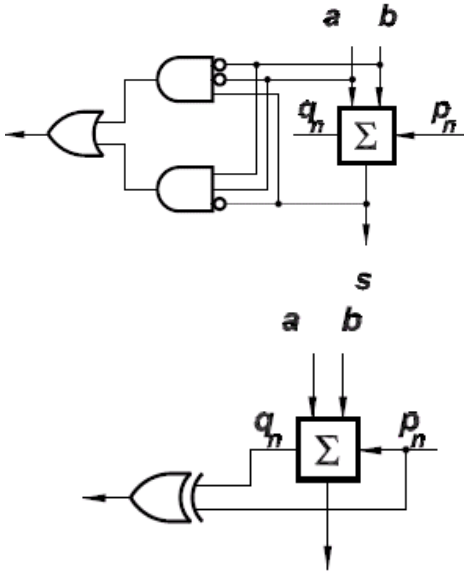
For 5 bit representation:



The incorrect result is smaller than each of addends

Overflow of Signed Binary Representation

- Result is incorrect, numeric value is outside of the range that can be represented with a given number of digits
- **It is manifested by result sign different from the sign of addends when both addends signs are the same**, and
- the exclusive-or (xor) of carry **to** and **from** MSB differs.



For 5 bit representation:

-4		1	1	1	0	0
+5	+	0	0	1	0	1
1	1	0	0	0	0	1

-4		1	1	1	0	0
-11	+	1	0	1	0	1
-15	1	1	0	0	0	1

12		0	1	1	0	0
+5	+	0	0	1	0	1
?-15	0	1	0	0	0	1

-4		1	1	1	0	0
-13	+	1	0	0	1	1
?15	1	0	1	1	1	1

Sign Extension

Example:

```
short int x =  15213;
int      ix = (int) x;
short int y = -15213;
int      iy = (int) y;
```

	Decimal	Hex	Binary
x	15213	3B 6D	00111011 01101101
ix	15213	00 00 C4 92	00000000 00000000 00111011 01101101
y	-15213	C4 93	11000100 10010011
iy	-15213	FF FF C4 93	11111111 11111111 11000100 10010011

Hardware Divider – Simple Sequential Algorithm

Non-restoring division

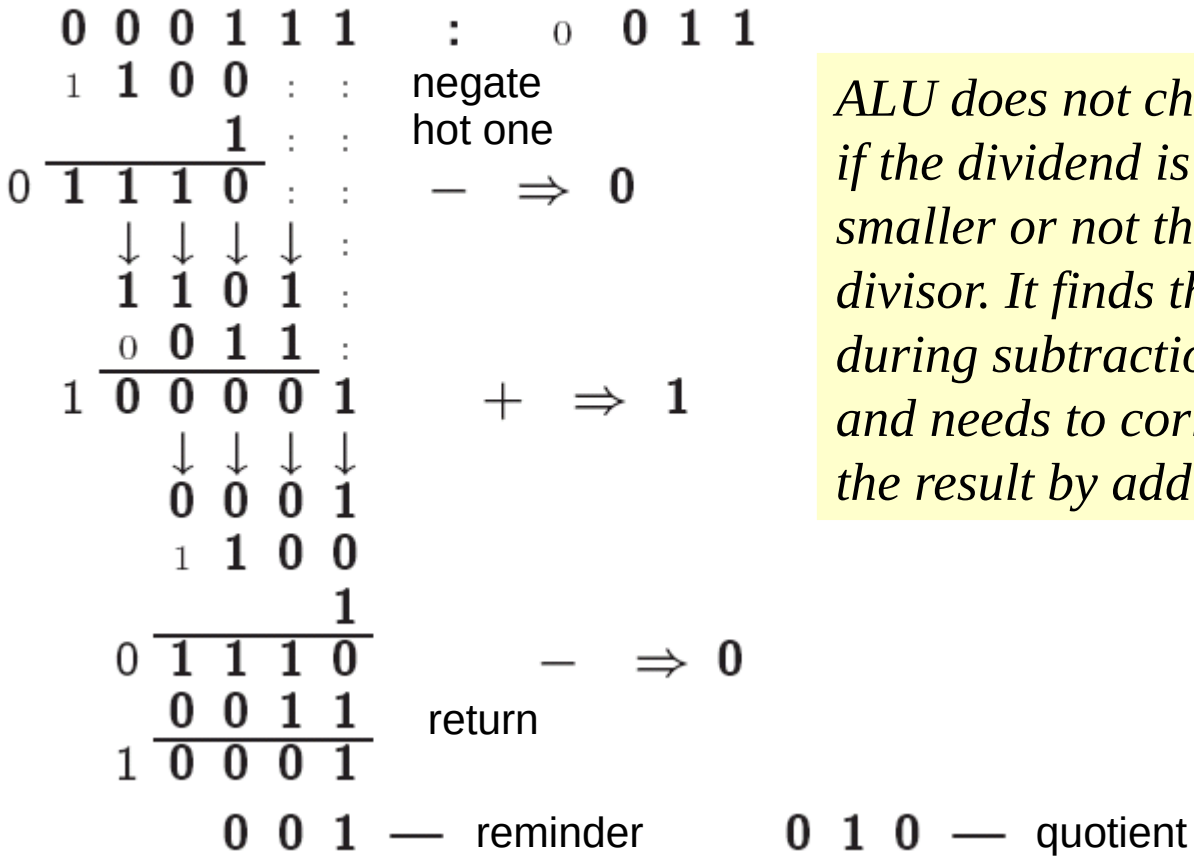
1 1 1 : 0 1 1

7 / 3

7 – 4*3 = -5

(non-restoring)
-5+2*3 = 1
=7 - 2*3

1 – 3 = -2
(restoring)
-2+3 = 1
Restoring is required only for last operation



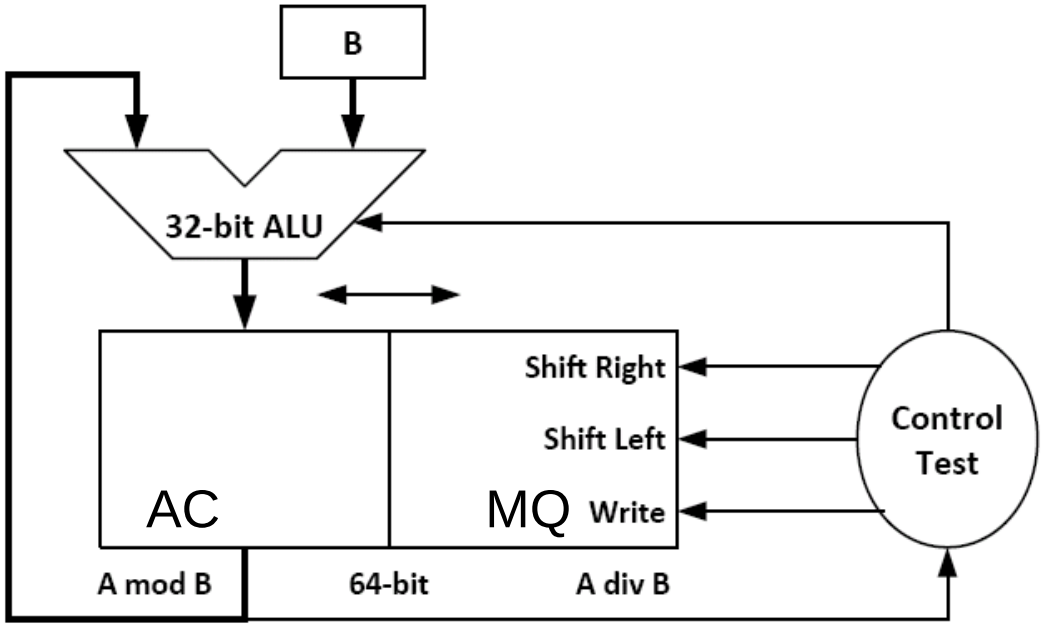
ALU does not check, if the dividend is smaller or not than divisor. It finds that during subtraction and needs to correct the result by addition.

Hardware divider logic (32b case)

1 1 1 : 0 1 1 dividend = quotient × divisor + reminder

```
  0 0 0 1 1 1   :   0 0 1 1
⊟   1 1 0 0 :   :   negate
        1 :   :   hot one
  0 1 1 1 0 :   :   − ⇒ 0
    ↓ ↓ ↓ ↓ :
    1 1 0 1 :
⊞   0 0 1 1 :
  1 0 0 0 0 1       + ⇒ 1
    ↓ ↓ ↓ ↓
    0 0 0 1
⊟     1 1 0 0
          1
  0 1 1 1 0         − ⇒ 0
N   0 0 1 1   return
  1 0 0 0 1
```

0 0 1 — reminder **0 1 0** — quotient



Algorithm of the sequential division

MQ = dividend;
B = divisor; (Condition: divisor is not 0!)
AC = 0;

```
for( int i=1; i <= n; i++)          {
   SL (shift AC MQ by one bit to the left, the LSB bit is kept on zero)
   if(AC >= B)   {
         AC = AC – B;
         MQ0 = 1;          // the LSB of the MQ register is set to 1
    }
}
```

→ Value of MQ register represents quotient and AC remainder

Example of X/Y division

Dividend x=1010 and divisor y=0011

i	operation	AC	MQ	B	comment
		0000	1010	0011	initial setup
1	**SL**	0001	0100		
	nothing	0001	0100		the if condition not true
2	**SL**	0010	1000		
		0010	1000		the if condition not true
3	**SL**	0101	0000		$r \geq y$
	AC = AC – B; MQ_0 = 1;	0010	0001		
4	**SL**	0100	0010		$r \geq y$
	AC = AC – B; MQ_0 = 1;	0001	0011		end of the cycle

x : y = 1010 : 0011 = 0011 reminder 0001, (10 : 3 = 3 reminder 1)

*Real Numbers

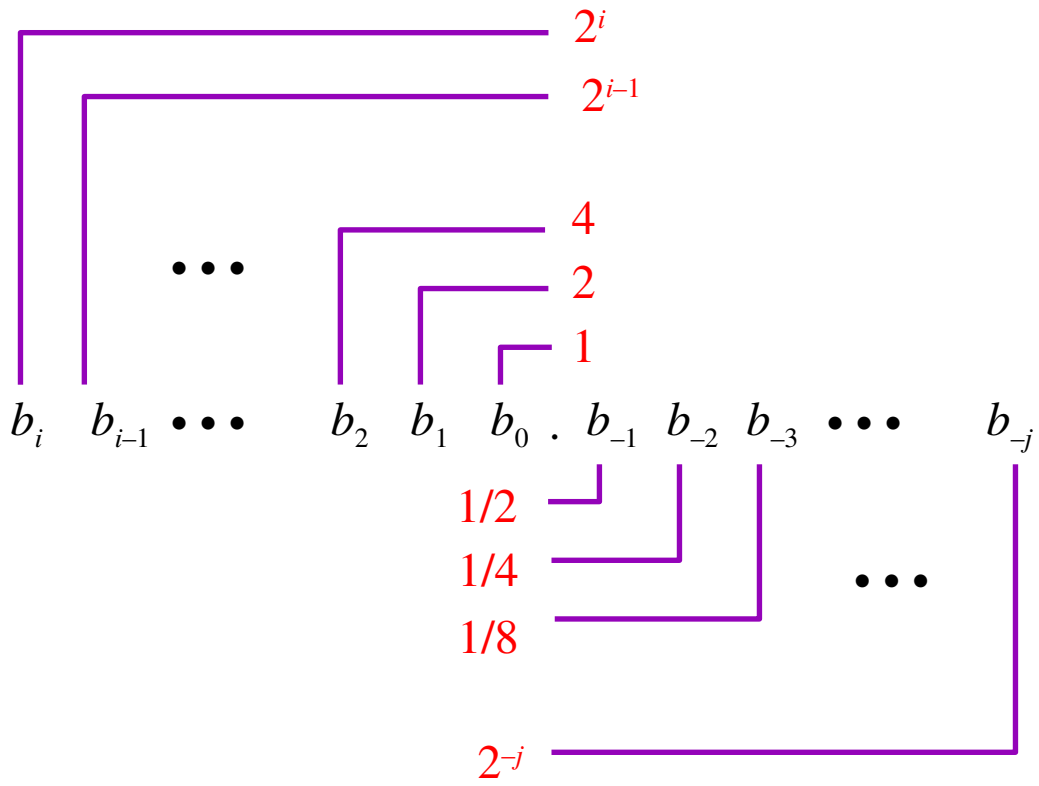
and their representation in computer

Higher Dynamic Range for Numbers (REAL/float)

- Scientific notation, semi-logarithmic, floating point
  - The value is represented by:
    - EXPONENT (E) – represents scale for given value
    - MANTISSA (M) – represents value in that scale
    - the sign(s) are usually separated as well
    - Mantissa x base$^{\text{Exponent}}$
- Normalized notation
  - The exponent and mantissa are adjusted such way, that mantissa is held in some standard range. Usually $\langle 1, \text{base})$
  - When considered base z=2 is considered then mantissa range is $\langle 1, 2)$ or alternatively $\langle 0.5, 1)$.
- Decimal representation: 7.26478×10^{3}
- Binary representation: $1{,}010011 \times 2^{1001}$

Fractional Binary Numbers/Fixed Point

They can be used directly or as base for mantissa of float



$$\sum_{k=-j}^{i} b_k \cdot 2^k$$

Real number representation in fixed point (fractional numbers)

Bits following “binary point” specify fractions in power two series

Fixed Point Examples

Value	*Representation*
5+3/4	101.11_2
2+7/8	10.111_2
63/64	0.111111_2

Operations

Divide by 2 → shift right

Multiply by 2 → shift left.

Numbers $0.111111\ldots_2$ are smaller than 1.0

$1/2 + 1/4 + 1/8 + \ldots + 1/2^i + \ldots \rightarrow 1.0$

Exact notation → $1.0 - \varepsilon$

Binary and Decimal Real Numbers Examples

$23.47 = 2\times10^{1} + 3\times10^{0} + 4\times10^{-1} + 7\times10^{-2}$

↑ decimal point

$10.01_{two} = 1\times2^{1} + 0\times2^{0} + 0\times2^{-1} + 1\times2^{-2}$

↑ binary point

$= 1\times2 + 0\times1 + 0\times\frac{1}{2} + 1\times\frac{1}{4}$

$= 2 + 0.25 = 2.25$

Scientific Notation and Binary Numbers

Decimal number:

-123 000 000 000 000 → -1.23×10^{14}

0.000 000 000 000 000 123 → $+1.23\times 10^{-16}$

Binary number:

110 1100 0000 0000 → $1.1011\times 2^{14} = 29696_{10}$

-0.0000 0000 0000 0001 1101 → -1.1101×2^{-16}

$=-2.765655517578125 \times 10^{-5}$

Standardized Format for REAL Type Numbers

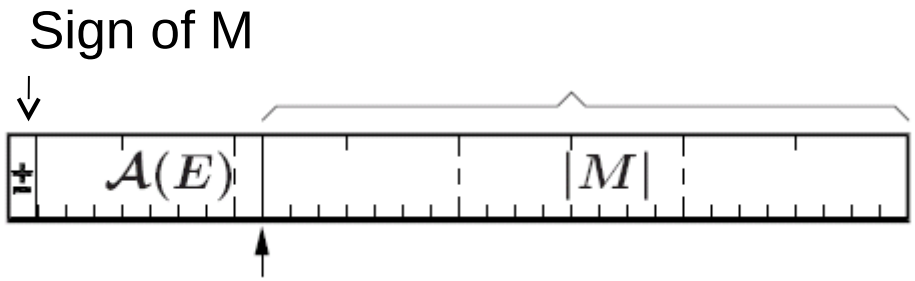
- Standard IEEE-754 defines next REAL representation and precision
  - single-precision – in the C language declared as **`float`**
    - uses 32 bits (1 + 8 + 23) to represent a number
  - double-precision – C language **`double`**
    - Uses 64 bits (1 + 11 + 52) to represent a number
  - actual standard (IEEE 754-2008) adds half-precision float (16 bits) mainly for graphics and neural networks, quadruple-precision (128 bits) and octuple-precision (256 bits) for special scientific computations

The Representation/Encoding of Floating Point Number

- Mantissa encoded as the sign and absolute value (magnitude) – equivalent to the **direct representation**
- Exponent encoded in **biased representation** (K=+127 for single precision, +1023 for double)
- The implicit leading one can be omitted due to normalization of $m \in \langle 1, 2)$ – 23+1 implicit bit for single

$$X = -1^{s}\ 2^{A(E)-127}\ m \qquad \text{where } m \in \langle 1, 2)$$

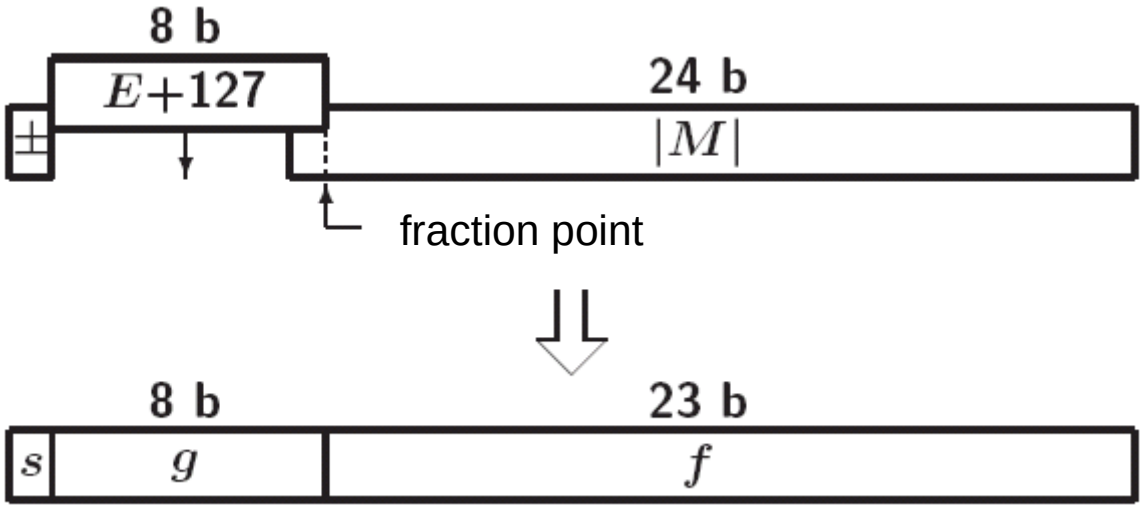
$$m = 1 + 2^{-23}\ M$$



Radix point position for E and M

ANSI/IEEE Std 754-1985 – 32b and 64b Formats

ANSI/IEEE Std 754-1985 — single precision format — 32b



ANSI/IEEE Std 754-1985 — double precision format — 64b

g . . . 11b *f* . . . 52b

ANSI/IEEE Std 754-1985 — half precision format — 16b

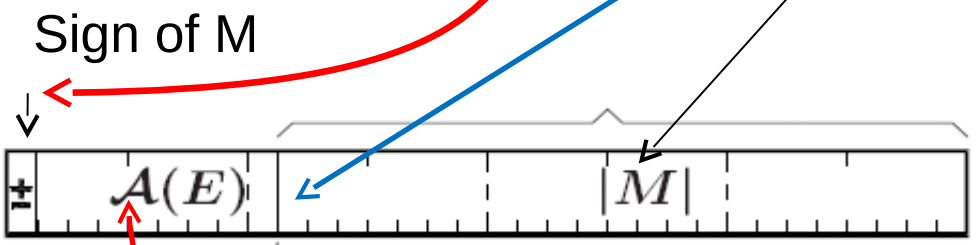
g . . . 5b *f* . . . 10b

Examples of (De)Normalized Numbers in Base 10 and 2

- -2.34×10^{56} ← normalized
- $+0.002 \times 10^{-4}$ ← not normalized
- $+987.02 \times 10^{9}$ ← not normalized

binary

- $\pm 1.xxxxxxx_2 \times 2^{yyyy}$



The radix point position for E and M

IEEE 754 – Conversion Examples

Find IEEE-754 float representation of -12.625_{10}

- Step #1: convert $-12.625_{10} = -1100.101_2 = 101 / 8$
- Step #2: normalize $-1100.101_2 = -1.100101_2 * 2^3$
- Step #3:

Fill sign field, negative for this case -> S=1.
Exponent + 127 -> 130 -> 1000 0010 .
The first mantissa bit 1 is a hidden one ->

1 1000 0010 . 1001 0100 0000 0000 0000 000

Alternative approach, separate sign, find floor of binary logarithm for absolute value, compute equivalent power of two, divide number (result is normalized) and, subtract one, multiply by two, if > 1 subtract and append 1 to result else append 0, multiply by two and repeat.

Example 0.75

$0.75_{10} = 0.11_{2} = 1.1 \times 2^{-1} = 3/4$

$1.1 = 1.\ F \rightarrow F = 1$

$E - 127 = -1 \rightarrow E = 127 - 1 = 126 = 01111110_{2}$

$S = 0$

$00111111010000000000000000000000 = 0x3F400000$

Example 0.1_{10} – Conversion to Float

$0.1_{10} = 0.000110011...._2$

$= 1.10011_2 \times 2^{-4} = 1.F \times 2^{E-127}$

F = 10011 -4 = E – 127

E = 127 -4 = 123 = 01111011_2

0011 1101 1100 1100 1100 1100 1100 1100 *1100 11..*

0x3DCCCCCD, why the last is a D ?

Example 0.1_{10} – Conversion to Float

$0.1_{10} = 0.\underline{0001100}11...._2 =$

```
0.0001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011001100110011001100110011001100110011001100110011001100110011001100110011
0011...
```

Often Inexact Floating Point Number Representation

Decadic number with finite expansion → infinite binary expansion

Examples:

0.1_{ten} → 0.2 → 0.4 → 0.8 → 1.6 → 3.2 → 6.4 → 12.8 →…

$0.1_{10} = 0.0001100110011001 1\ldots_2$

$= 0.0\underline{0011}_2$ (infinite bit stream)

More bits only enhance precission of 0.1_{10} representation

Real Number Representation - Limitations

Limitation

Only numbers corresponding to $x/2^k$ allows exact representation, all other are stored inexact

Value representation

1/3 $0.0101010101[01]\ldots_2$

1/5 $0.0011001100 11[0011]\ldots_2$

1/10 $0.0001100110011[0011]\ldots_2$

Special Values – Not a Number (NaN) and Infinity

- If the result of the mathematical operation is not defined, such as the calculation of log (-1), or the result is ambiguous 0/0, +Inf + -Inf, then the value NaN (Not-a-Number) is saved

  = exponent is set to all ones and the mantissa is nonzero.

NaN

positive	**0 11111111 *mantisa !=0***	**NaN**

- If the operation results only overflow the range or infinity is on input (X + +Inf) and result sign is unambiguous

Infinity

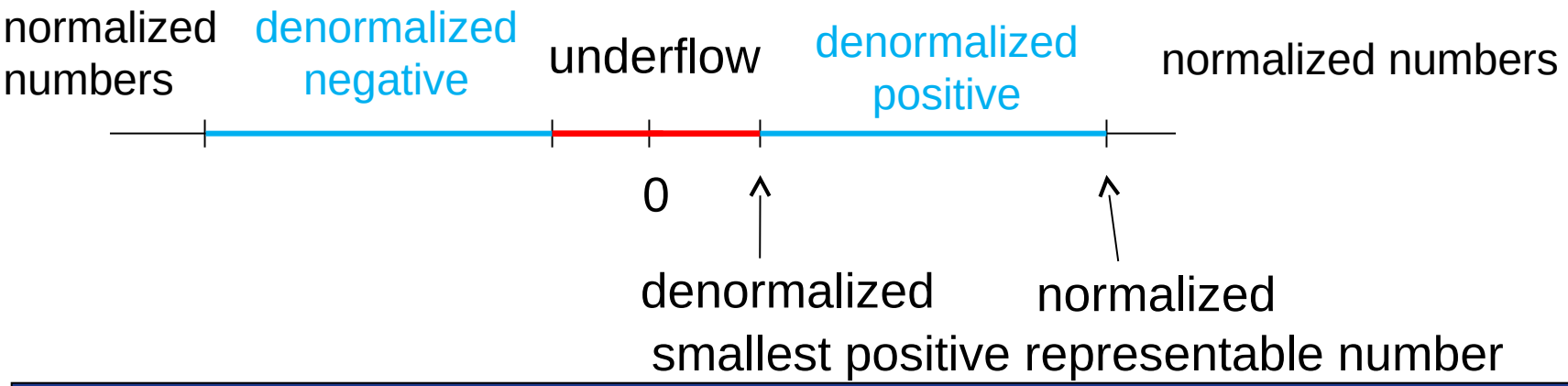
positive	**0 11111111 00000000000000000000000**	**+Inf**
negative	**1 11111111 00000000000000000000000**	**-Inf**

Implied (Hidden) Leading 1 bit

- Most significant bit of the mantissa is one for each normalized number and it is not stored in the representation for the normalized numbers
- If exponent representation is zero then encoded value is zero or denormalized number which requires to store most significant bit and there is zero considered on usual hidden one location
- Denormalized numbers allow to keep resolution in the range from the smallest normalized number to zero but the computation when some of operands is denormalized is more complex. Some coprocessors do not support denormalized numbers and emulation is required to fulfill IEEE-754 strict requirements, Intel coprocessors supports denormalized numbers

Underflow/Lost of the Precision for IEEE-754 Representation

- The case where stored number value is not zero but it is smaller than smallest number which can be represented in the normalized form
- The direct underflow to the zero can be prevented by extension of the representation range by denormalized numbers



Representation of the Fundamental Values

Zero

Positive zero	**0 00000000 00000000000000000000000**	**+0.0**
Negative zero	**1 00000000 00000000000000000000000**	**-0.0**

Infinity

Positive infinity	**0 11111111 00000000000000000000000**	**+Inf**
Negative infinity	**1 11111111 00000000000000000000000**	**-Inf**

Representation corner values

Smallest normalized	*** 00000001 00000000000000000000000**	$\pm 2^{(1-127)}$ $\pm 1.1755\ 10^{-38}$
Biggest denormalized	*** 00000000 11111111111111111111111**	$\pm(1-2^{-23})2^{(1-126)}$
Smallest denormalized	*** 00000000 00000000000000000000001**	$\pm 2^{-23}2^{-126}$ $\pm 1.4013\ 10^{-45}$
Max. value	**0 11111110 11111111111111111111111**	$(2-2^{-23})2^{(127)}$ **+3.4028 10^{+38}**

The Table in Another Format

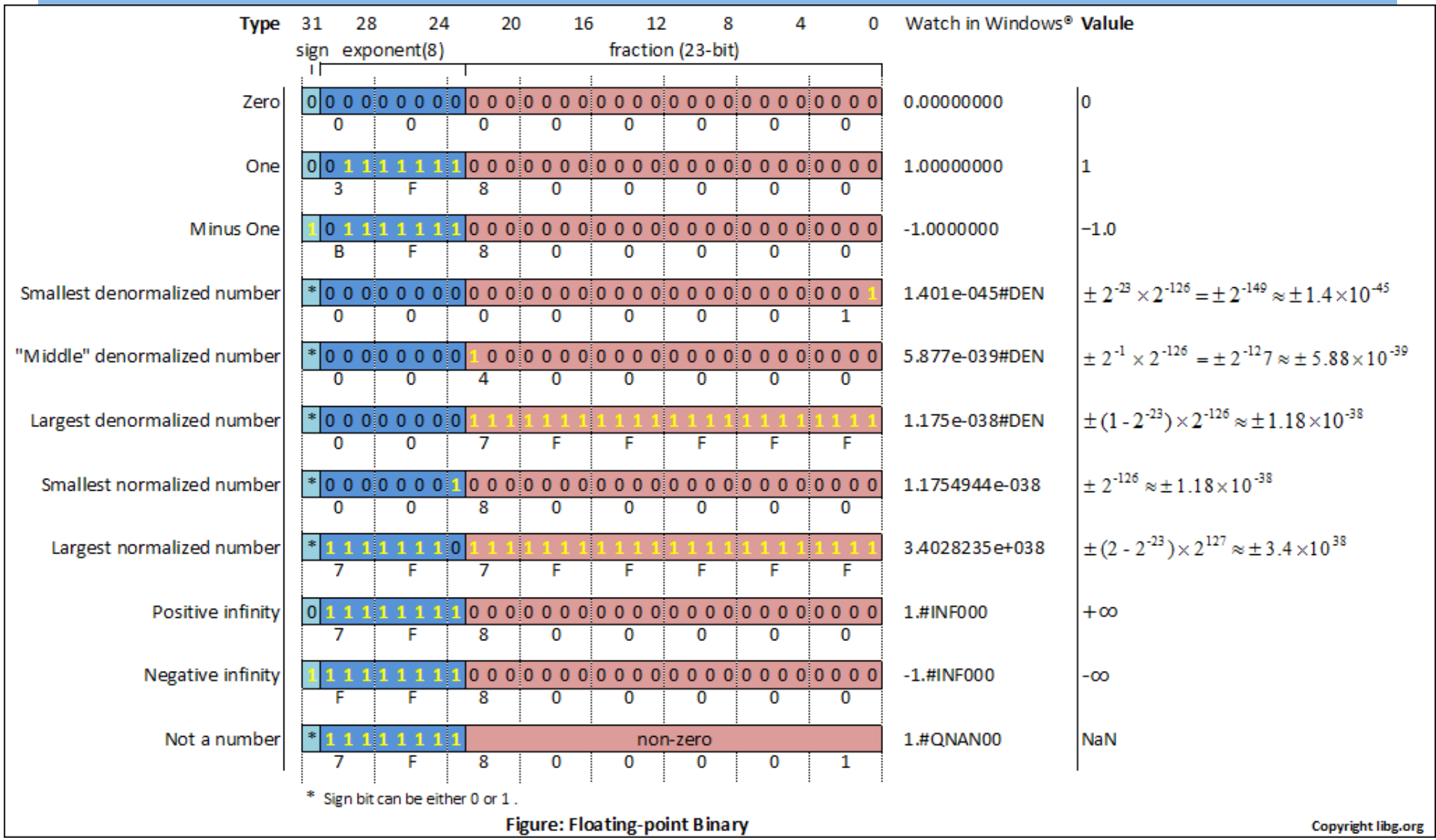


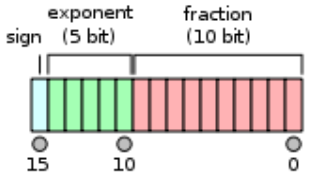
Figure: Floating-point Binary

Some Features of ANSI/IEEE Standard Floating-point Formats

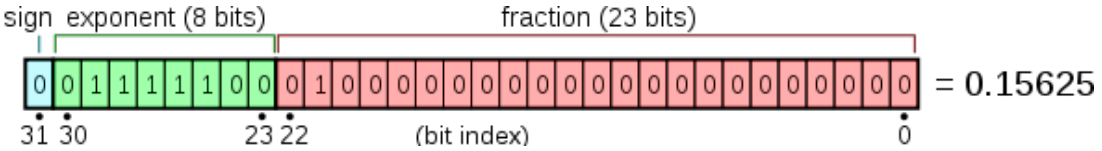
Feature	Single/Float	Double/Long
Word width in bits	32	64
Significand in bits	23 + 1 hidden	52 + 1 hidden
Significand range	$[1, 2 - 2^{-23}]$	$[1, 2 - 2^{-52}]$
Exponent bits	8	11
Exponent bias	127	1023
Zero (±0)	e + bias = 0, f = 0	e + bias = 0, f = 0
Denormal	e + bias = 0, $f \neq 0$ represents $\pm 0.f \times 2^{-126}$	e + bias = 0, $f \neq 0$ represents $\pm 0.f \times 2^{-1022}$
Infinity (±∞)	e + bias = 255, f = 0	e + bias = 2047, f = 0
Not-a-number (NaN)	e + bias = 255, $f \neq 0$	e + bias = 2047, $f \neq 0$
Ordinary number	e + bias $\in [1, 254]$ $e \in [-126, 127]$ represents $1.f \times 2^e$	e + bias $\in [1, 2046]$ $e \in [-1022, 1023]$ represents $1.f \times 2^e$
min	$2^{-126} \cong 1.2 \times 10^{-38}$	$2^{-1022} \cong 2.2 \times 10^{-308}$
max	$\cong 2^{128} \cong 3.4 \times 10^{38}$	$\cong 2^{1024} \cong 1.8 \times 10^{308}$

IEEE-754 Formats

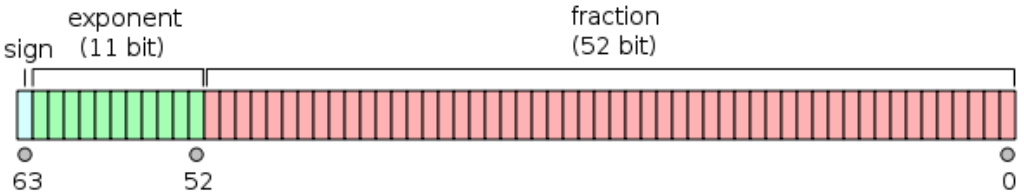
Half precision (binary16)



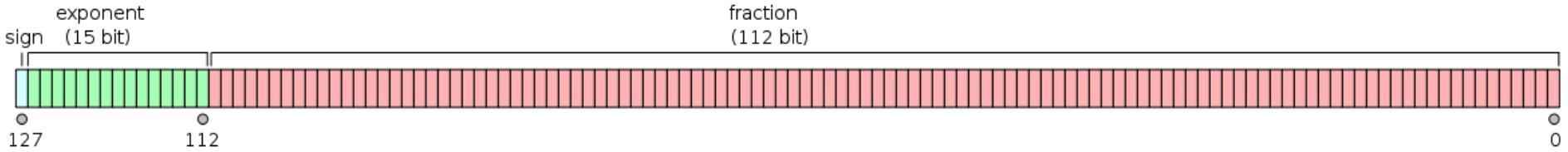
Single precision (binary32)



Double precision (binary64)

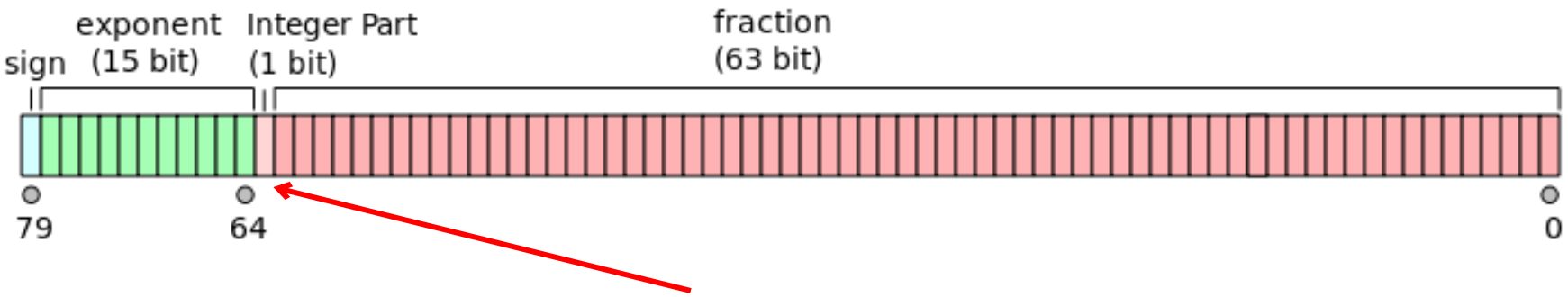


Quadruple precision (binary128)



Source: Herbert G. Mayer, PSU

X86 Extended Precision Format (80-bits)



Bit 1. is not hidden in mantissa!

Advanced readers note:

- *Intel processors integrate arithmetic coprocessor on the single chip with processor (from Intel 80486), which computes float and double expressions in „extended precision“ internally and the results are rounded to float/double when stored.*
- *But Streaming SIMD Extensions (SSE) instructions (vector operations) from Intel Pentium III on provides only double precision and the result rounding/precission* can be dependent on compiler selection

Source: Herbert G. Mayer, PSU

IEEE-754 Special Values Summary

sign bit	Exponent representation	Mantissa	Represented value/meaning
0	0<e<255	any value	normalized positive number
1	0<e<255	any value	normalized negative number
0	0	>0	denormalized positive number
1	0	>0	denormalized negative number
0	0	0	positive zero
1	0	0	negative zero
0	255	0	positive infinity
1	255	0	negative infinity
0	255	$\neq 0$	NaN – does not represent a number
1	255	$\neq 0$	NaN – does not represent a number

Comparison

- Comparison of the two IEEE-754 encoded numbers requires to solve signs separately but then it can be processed by unsigned ALU unit on the representations

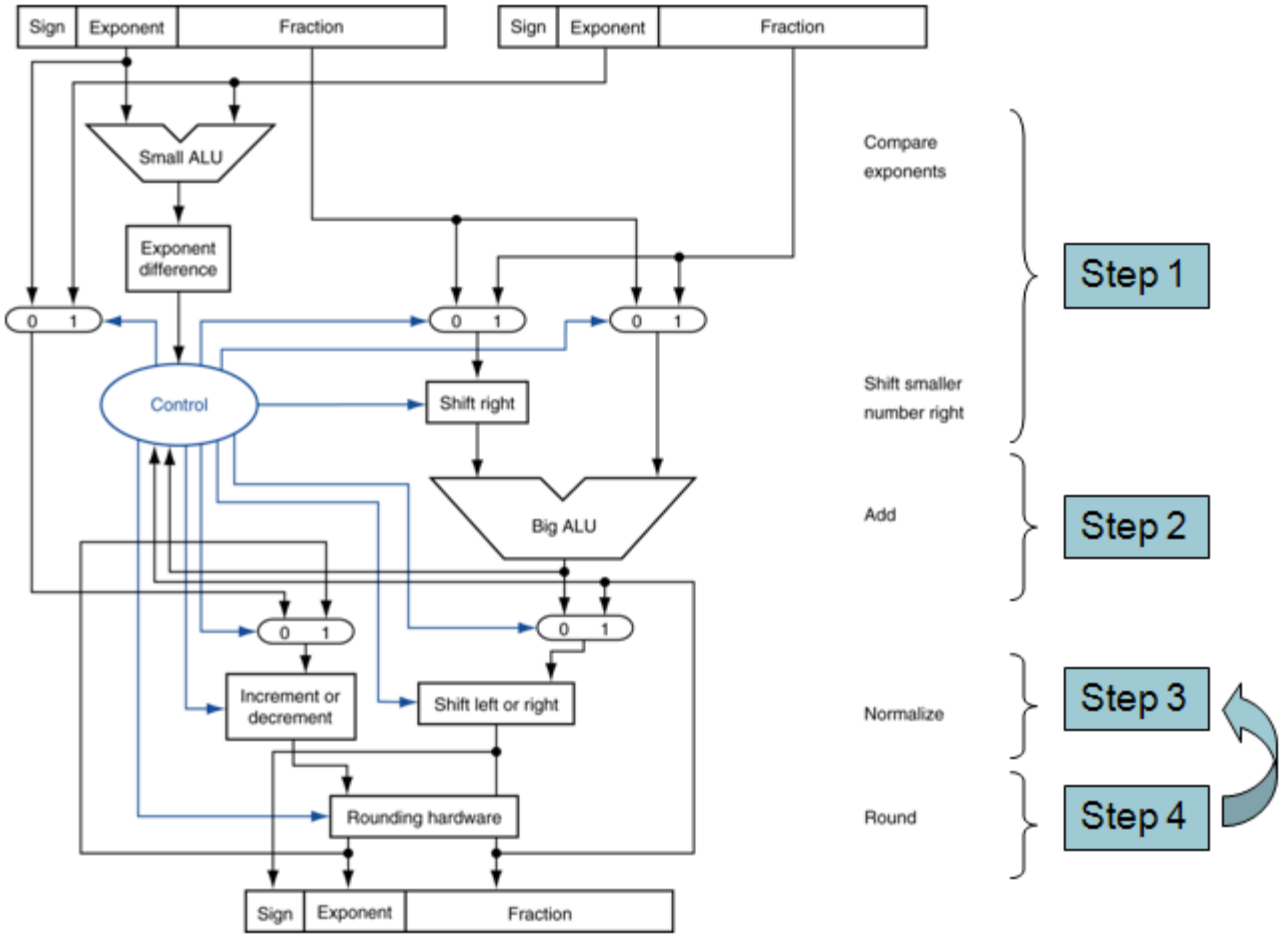
$$A \geq B \Longleftrightarrow A - B \geq 0 \Longleftrightarrow D(A) - D(B) \geq 0$$

- This is advantage of the selected encoding and reason why sign is not placed at start of the mantissa

Addition of Floating Point Numbers

- The number with bigger exponent value is selected
- Mantissa of the number with smaller exponent is shifted right – the mantissas are then expressed at same scale
- The signs are analyzed and mantissas are added (same sign) or subtracted (smaller number from bigger)
- The resulting mantissa is shifted right (max by one) if addition overflows or shifted left after subtraction until all leading zeros are eliminated
- The resulting exponent is adjusted according to the shift
- Result is normalized after these steps
- The special cases and processing is required if inputs are not regular normalized numbers or result does not fit into normalized representation

Hardware of the Floating Point Adder



Multiplication of Floating Point Numbers

- Exponents are added and signs xor-ed
- Mantissas are multiplied
- Result can require normalization

  max 2 bits right for normalized numbers
- The result is rounded

- Hardware for multiplier is of the same or even lower complexity as the adder hardware – only adder part is replaced by unsigned multiplier

Floating Point Arithmetic Operations Overview

Addition: $A \cdot z^a$, $B \cdot z^b$, $b < a$ — unify exponents

$B \cdot z^b = (B \cdot z^{b-a}) \cdot z^{b-(b-a)}$ — by shift of mantissa

$A \cdot z^a + B \cdot z^b = [A+(B \cdot z^{b-a})] \cdot z^a$ — sum + normalization

Subtraction: unification of exponents, subtraction and normalization

Multiplication: $A \cdot z^a \cdot B \cdot z^b = A \cdot B \cdot z^{a+b}$

$A \cdot B$ - normalize if required

$A \cdot B \cdot z^{a+b} = A \cdot B \cdot z \cdot z^{a+b-1}$ - by left shift

Division: $A \cdot z^a / B \cdot z^b = A/B \cdot z^{a-b}$

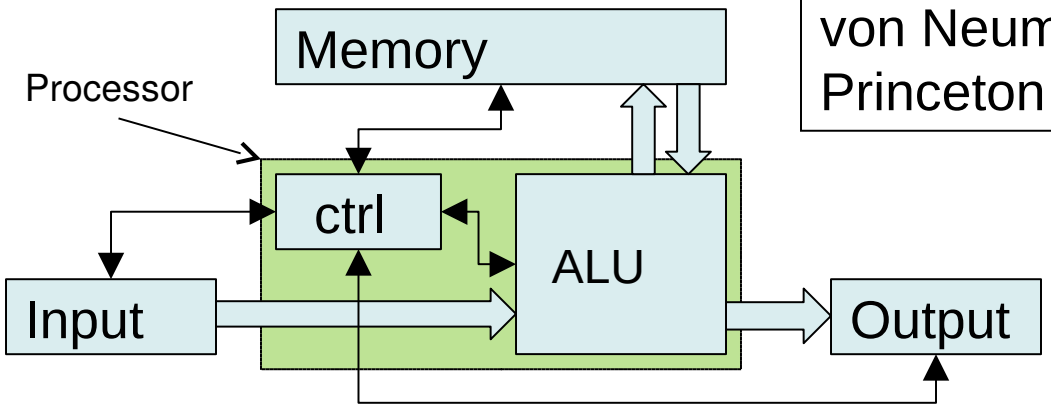
A/B - normalize if required

$A/B \cdot z^{a-b} = A/B \cdot z \cdot z^{a-b+1}$ - by right shift

*Memory and Data

and their store in computer memory

John von Neumann Computer Block Diagram



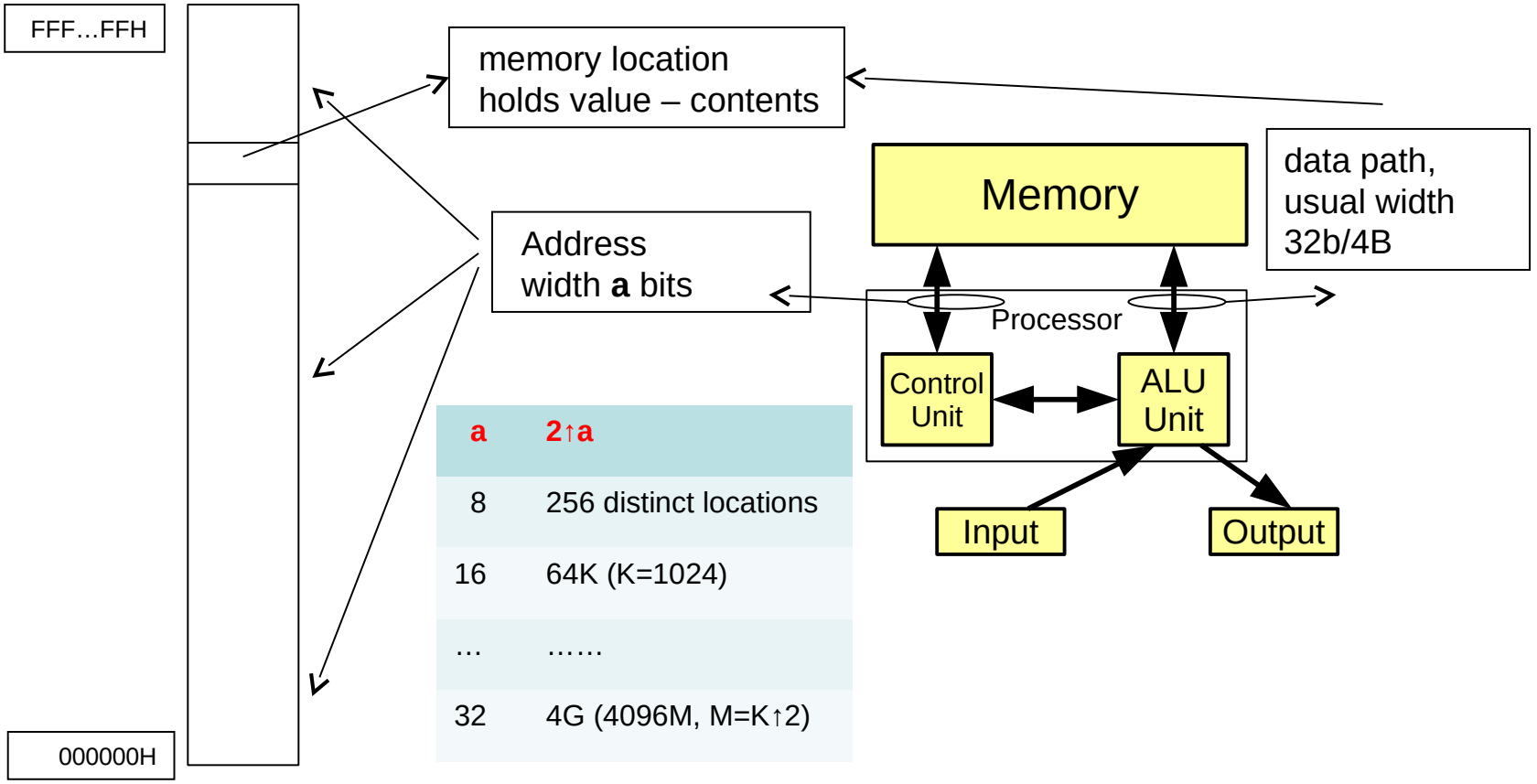
von Neumann's computer architecture
Princeton Institute for Advanced Studies

28. 12. 1903 -
8. 2. 1957

- 5 functional units – control unit, arithmetic logic unit, memory, input (devices), output (devices)
- An computer architecture should be independent of solved problems. It has to provide mechanism to load program into memory. The program controls what the computer does with data, which problem it solves.
- Programs and results/data are stored in the same memory. That memory consists of a cells of same size and these cells are sequentially numbered (address).
- The instruction which should be executed next, is stored in the cell exactly after the cell where preceding instruction is stored (exceptions branching etc.).
- The instruction set consists of arithmetics, logic, data movement, jump/branch and special/control instructions.

Memory Address Space

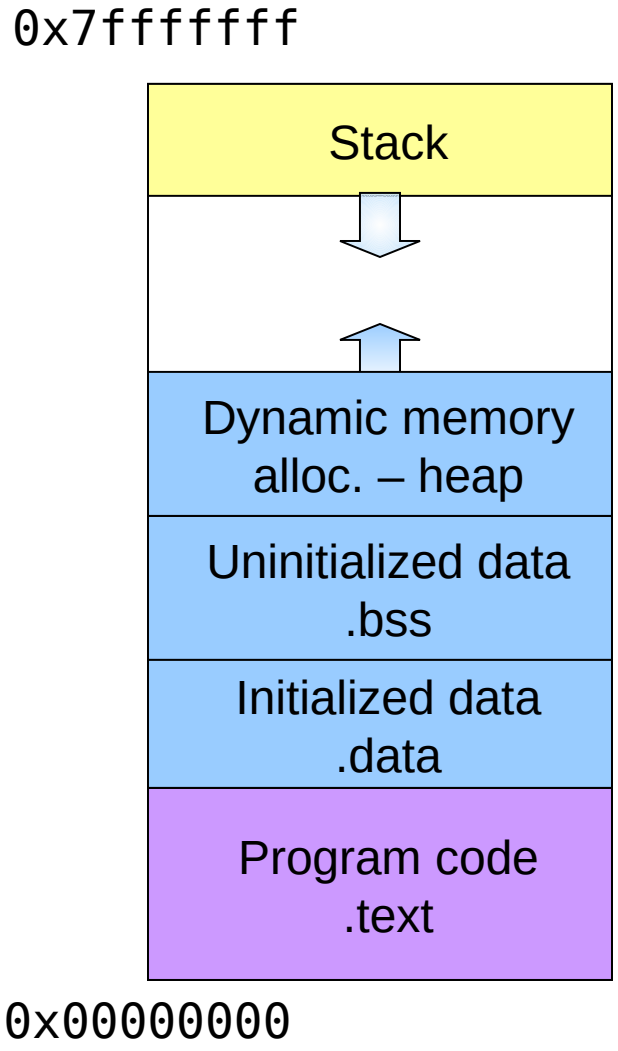
It is an array of addressable units (locations) where each unit can hold a data value.
Number/range of addresses same as addressable units/words are limited in size.



a	2↑a
8	256 distinct locations
16	64K (K=1024)
…	……
32	4G (4096M, M=K↑2)

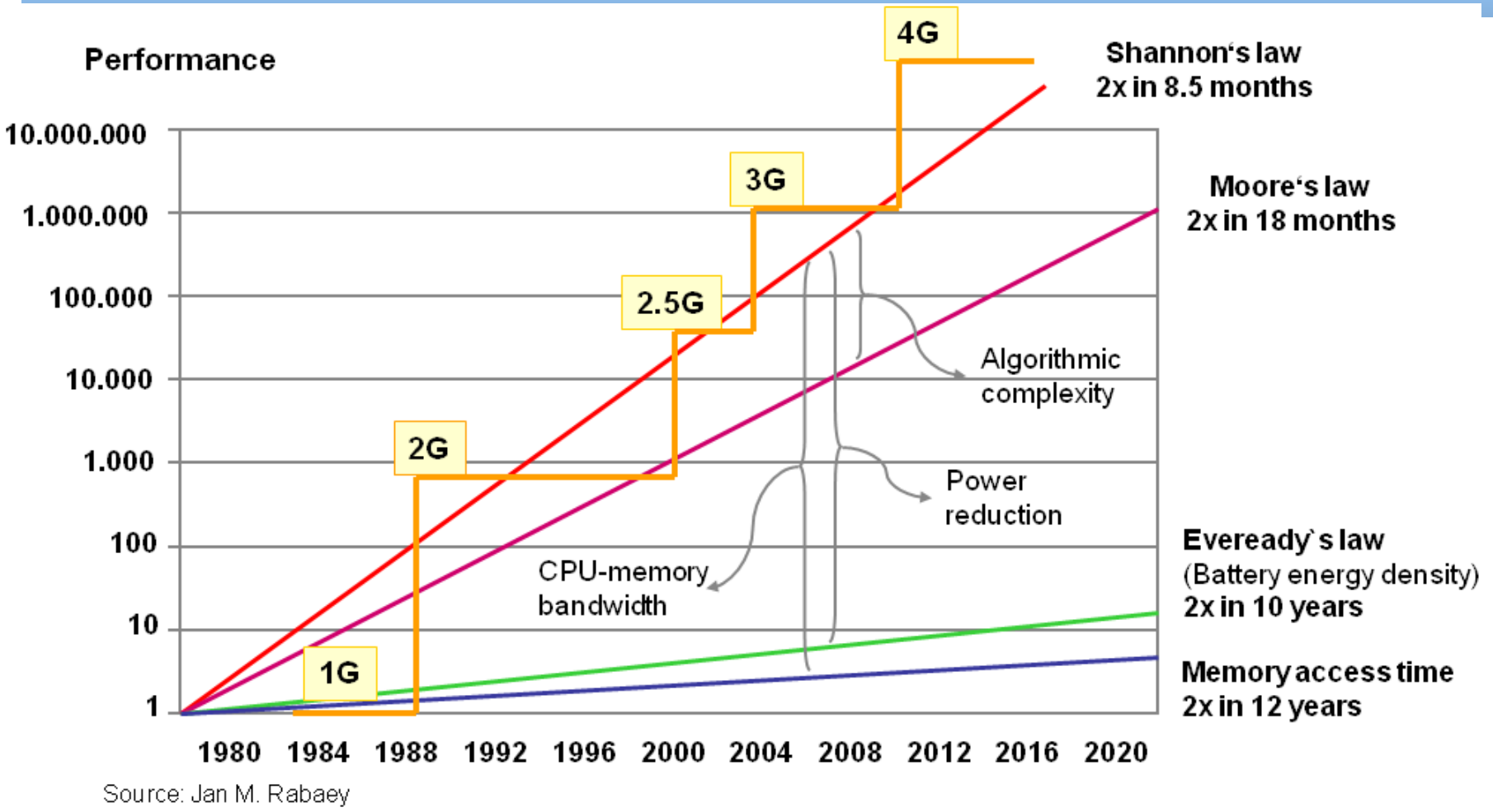
The most common size of addressable memory unit is 1B (8 bits)

Program Layout in Memory at Process Startup



- The executable file is mapped ("loaded") to process address space – sections **.data** and **.text** (note: LMA != VMA for some special cases)
- Uninitialized data area (**.bss –** block starting by symbol) is reserved and zeroed for C programs
- Stack pointer is set and control is passed to the function _**start**
- Dynamic memory is usually allocated above _**end** symbol pointing after **.bss**

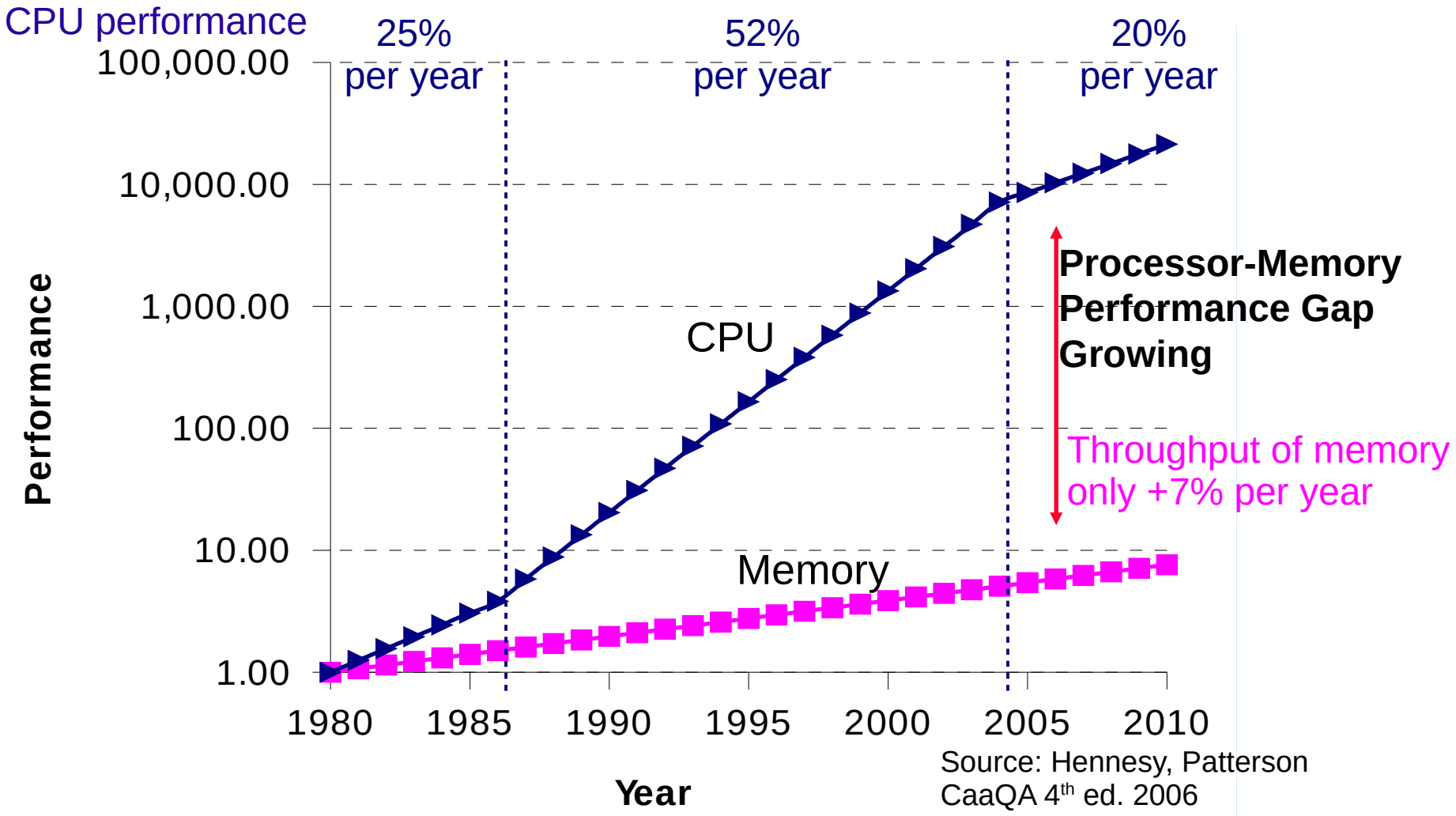
Key Technological Gaps Prediction



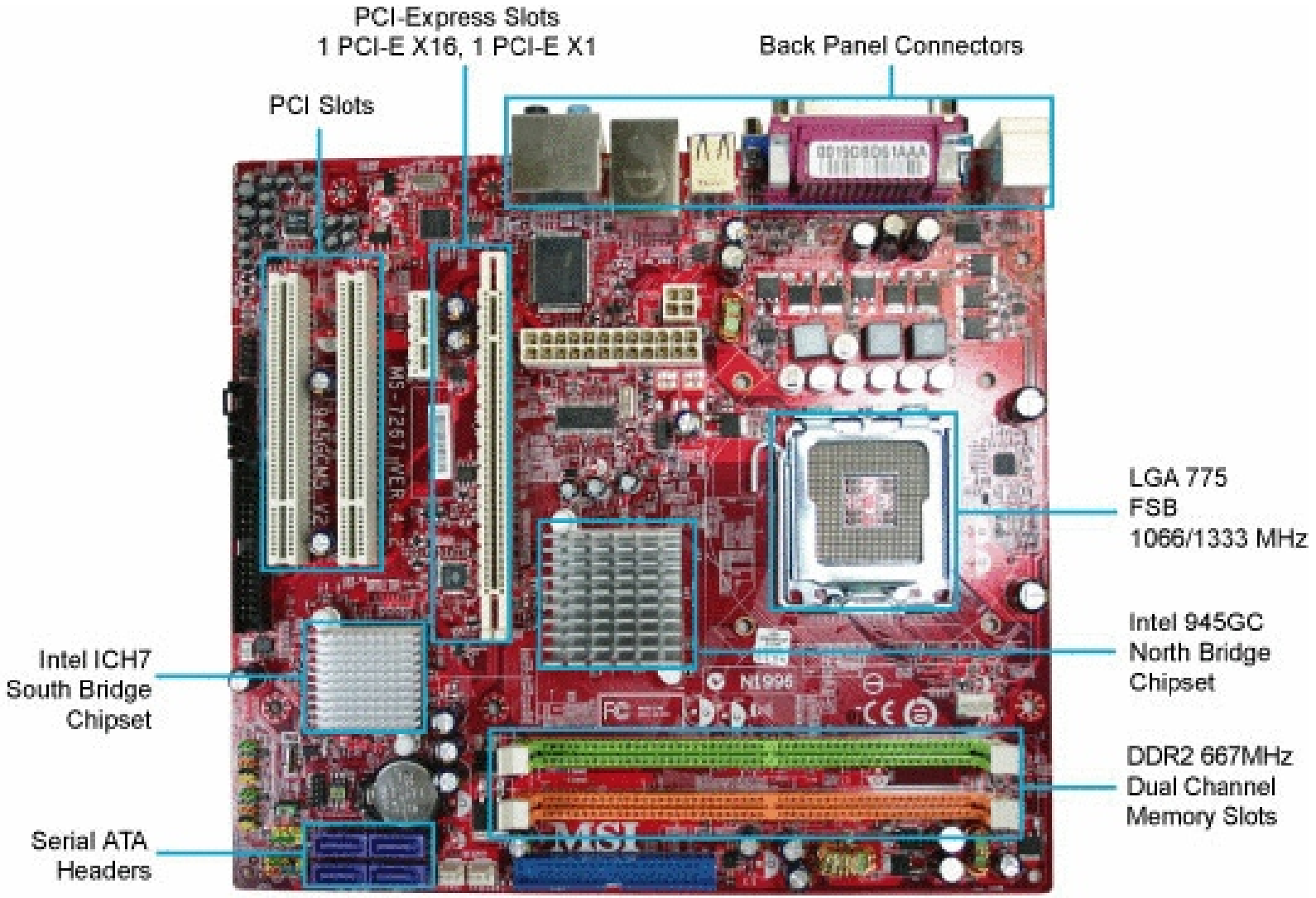
Source: Jan M. Rabaey

Note: The increase in complexity of algorithms over time has been formalized in literature as the so-called Shannon's Law of Algorithmic Complexity.

Memory and CPU Speed – Moore's Law



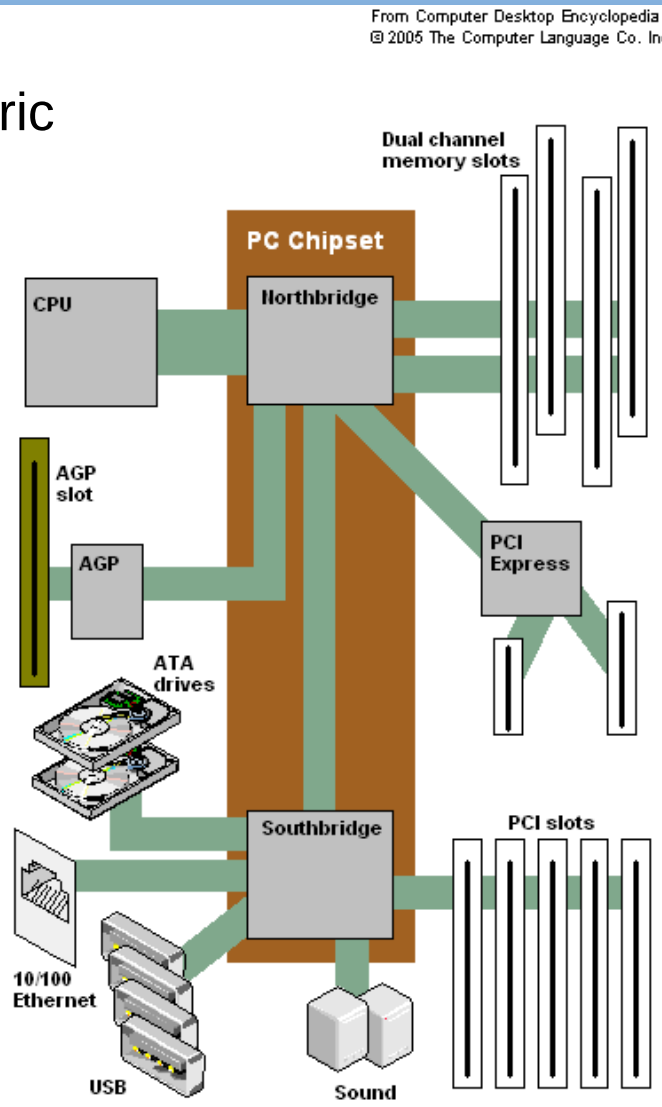
PC Computer Motherboard



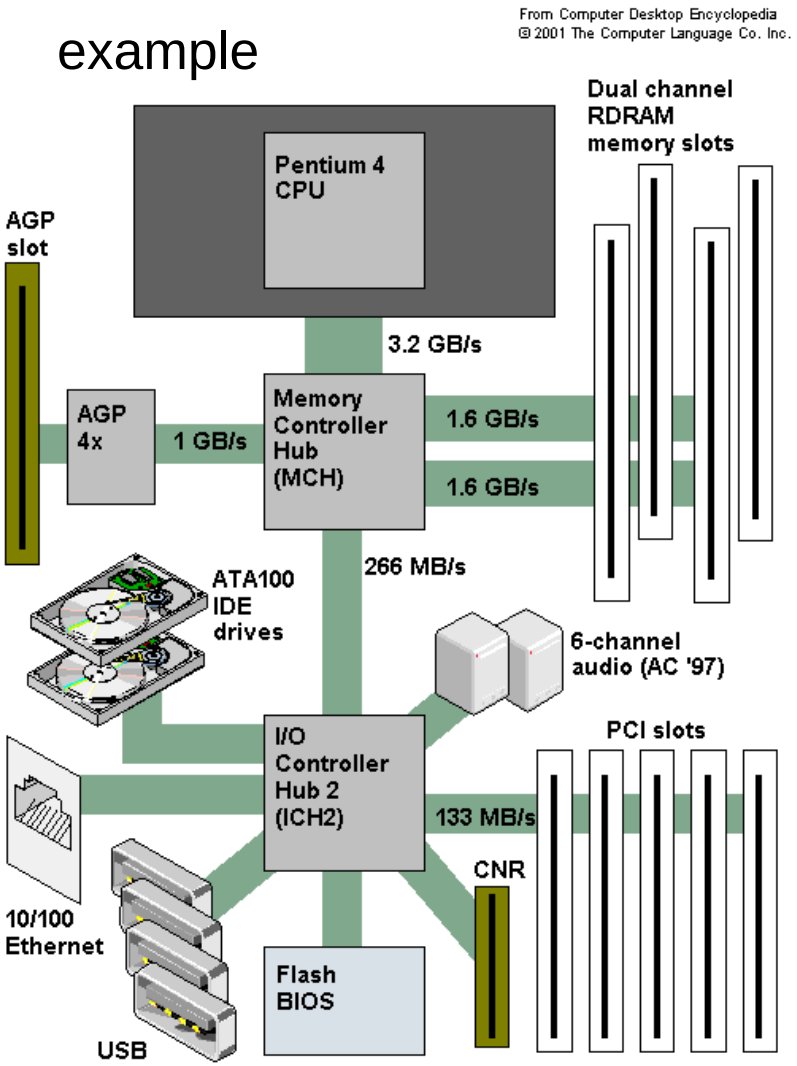
http://www.pcplanetsystems.com/abc/product_details.php?

Computer Architecture (Desktop x86 PC)

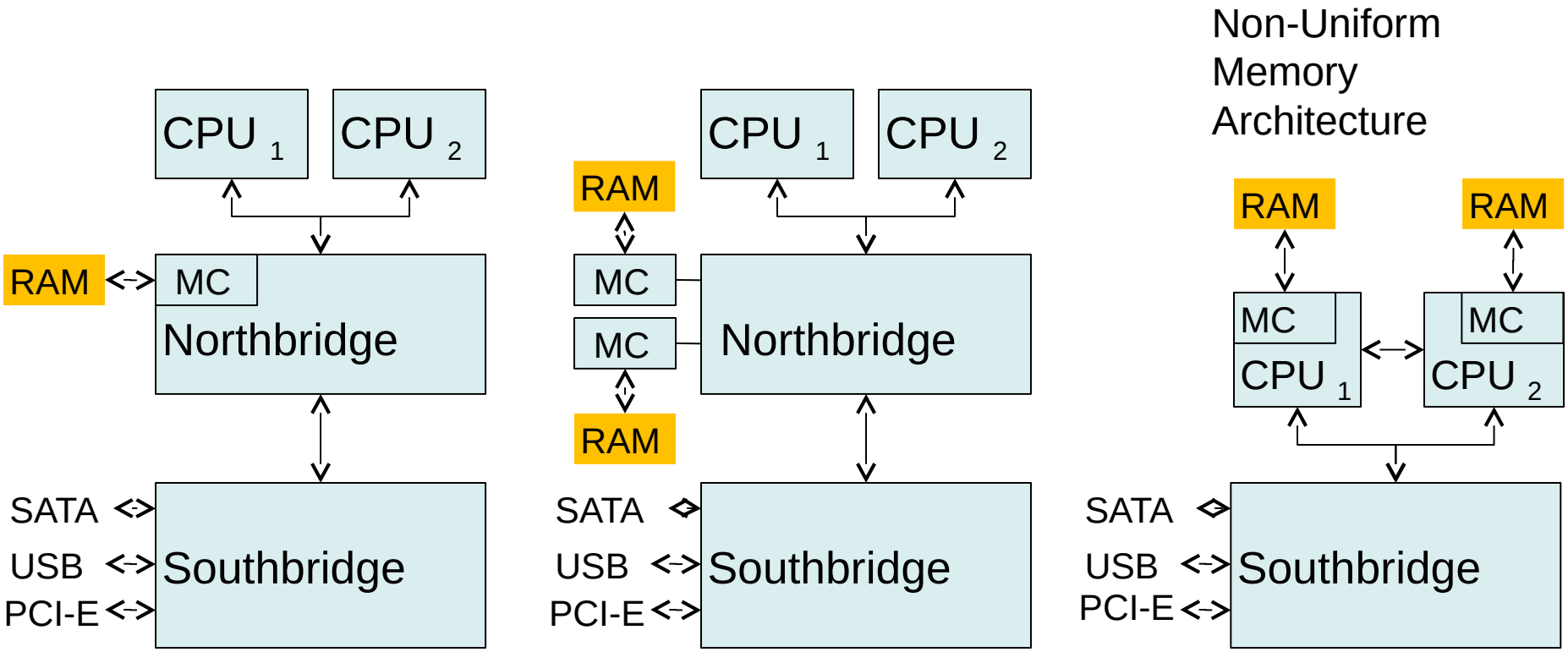
generic



example

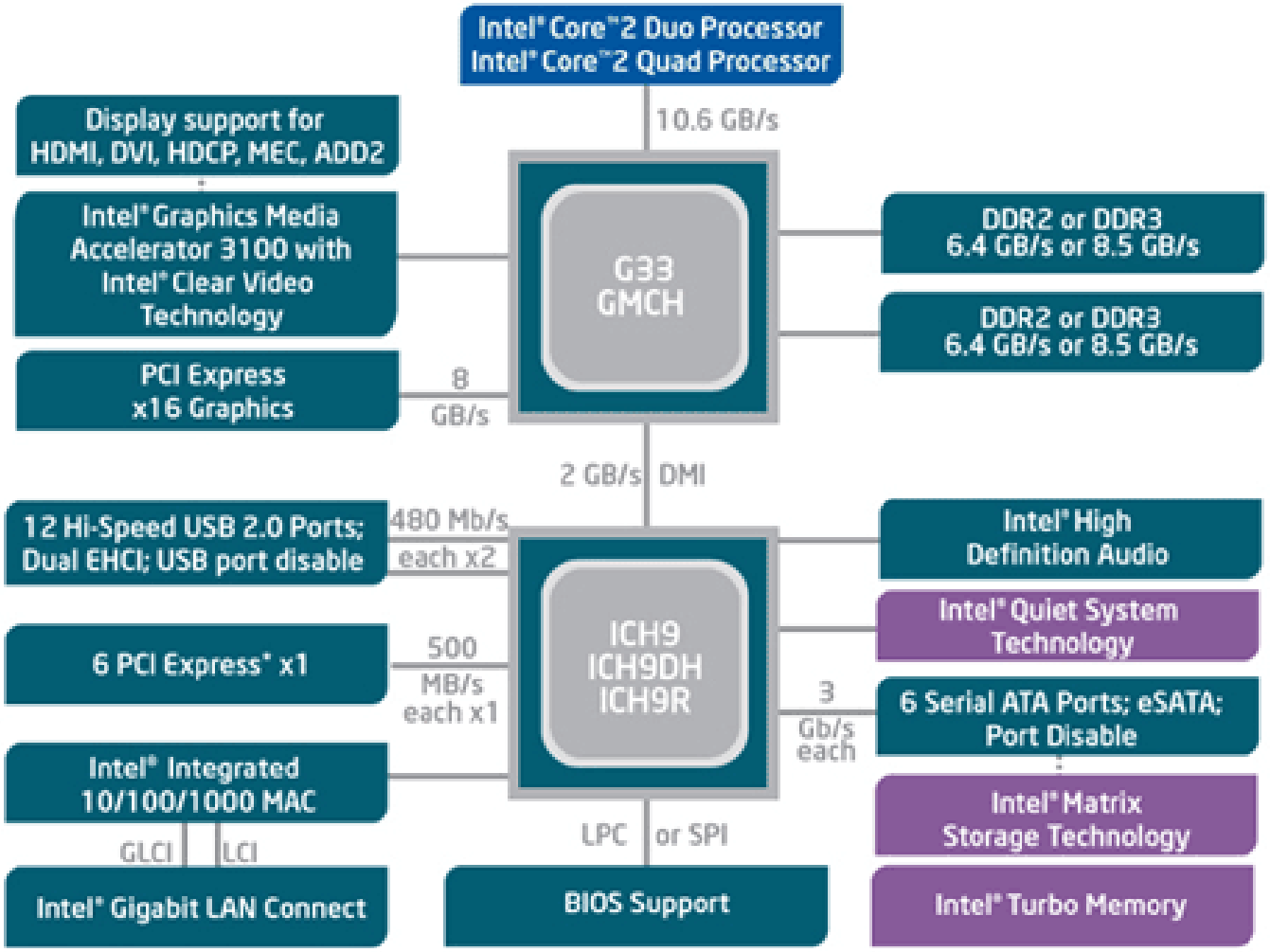


From UMA to NUMA Development (Even in PC Segment)



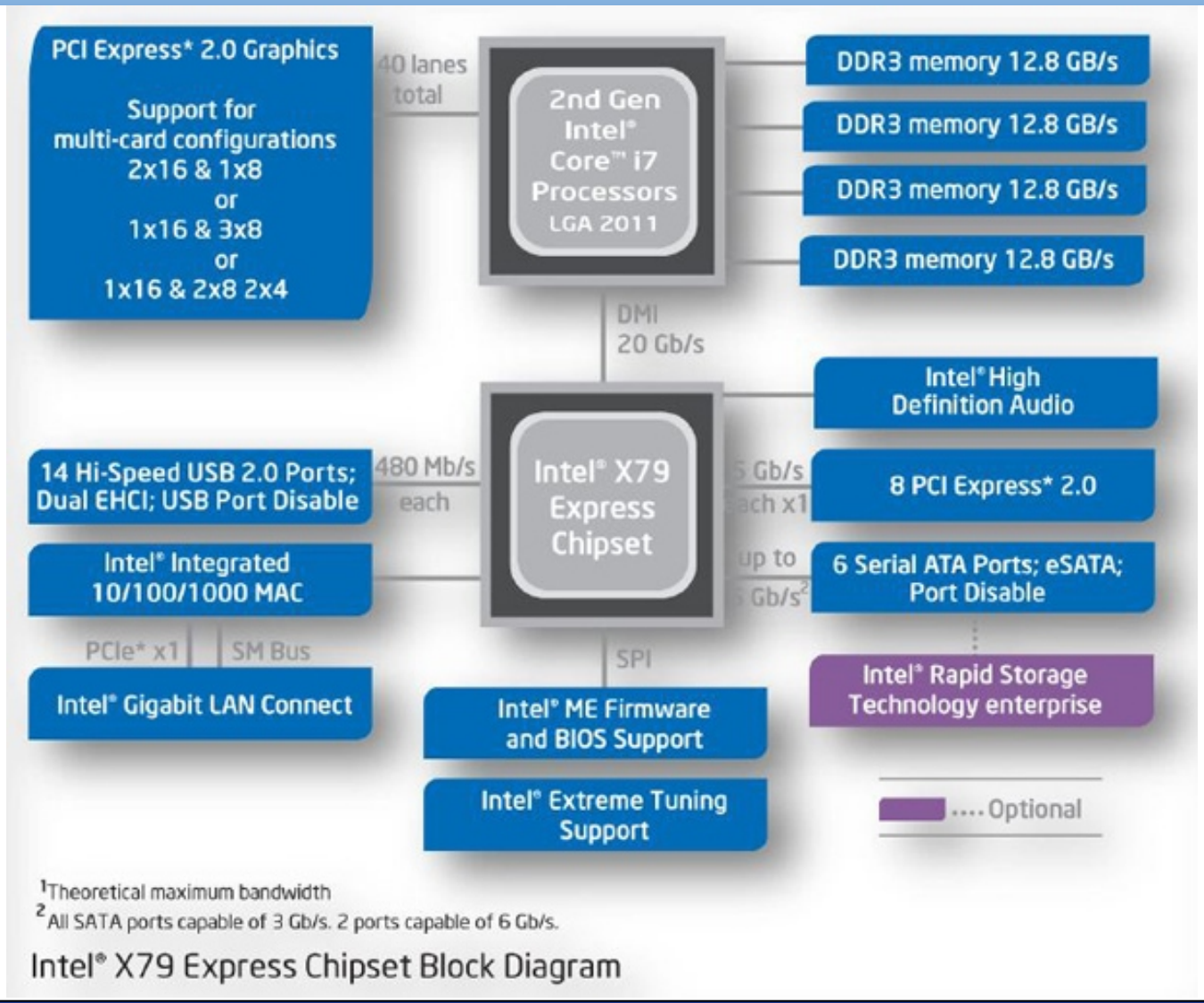
MC - Memory controller – contains circuitry responsible for SDRAM read and writes. It also takes care of refreshing each memory cell every 64 ms.

Intel Core 2 Generation



Northbridge became Graphics and Memory Controller Hub (GMCH)

Intel i3/5/7 Generation



Intel® X79 Express Chipset Block Diagram

Memory Subsystem – Terms and Definitions

- Memory address – fixed-length sequences of bits or index
- Data value – the visible content of a memory location

  Memory location can hold even more control/bookkeeping information
  - validity flag, parity and ECC bits etc.

- Basic memory parameters:
  - Access time – delay or latency between a request and the access being completed or the requested data returned
  - Memory latency – time between request and data being available (does not include time required for refresh and deactivation)
  - Throughput/bandwidth – main performance indicator. Rate of transferred data units per time.
  - Maximal, average and other latency parameters

Memory Types and Maintenance

- Types: RWM (RAM), ROM, FLASH
- Implementation: SRAM, DRAM

- Data retention time and conditions (volatile/nonvolatile)
- Dynamic memories (DRAM, SDRAM) require specific care
  - Memory refresh – state of each memory cell has to be internally read, amplified and fed back to the cell once every refresh period (usually about 60 ms), even in idle state. Each refresh cycle processes one row of cells.
  - Precharge – necessary phase of access cycle to restore cell state after its partial discharge by read
  - Both contribute to maximal and average access time.

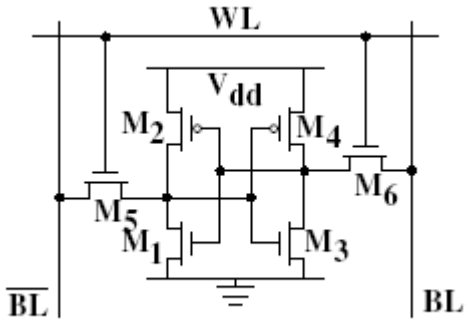
Typical Memory Parameters

- Memory types: RWM (RAM), ROM, FLASH,
- RAM realization:
  SRAM (static), **DRAM** (dynamic).

- RAM = *Random Access Memory*

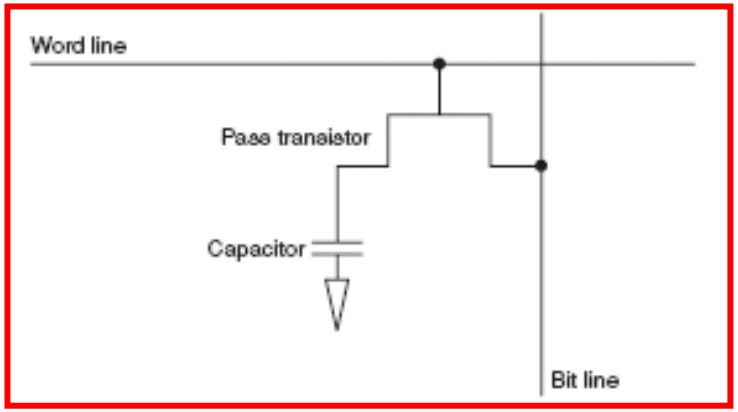
type	transistors per cell	1 bit area	data availability	latency
SRAM	cca 6	< 0,1 μm^2	always	< 1ns – 5ns
DRAM	1	< 0,001 μm^2	requires refresh	today 20 ns – 35 ns

Detail of static and Dynamic Memory Bit Cell



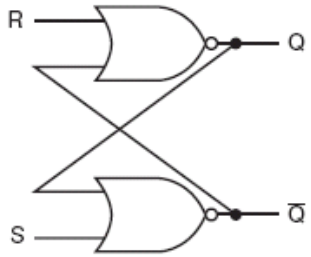
6 transistor static memory cell (single bit)

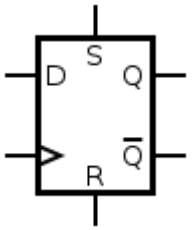
Single transistor cell of dynamic memory



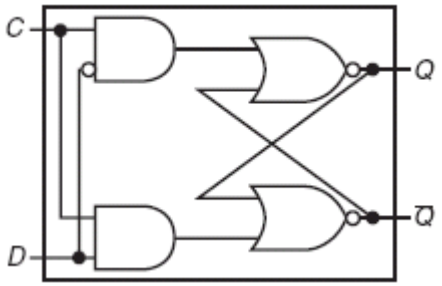
Flip-flop Circuits

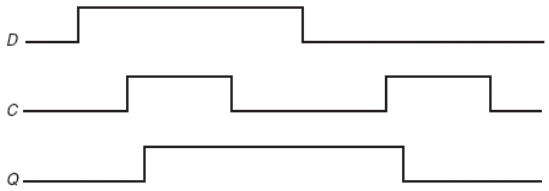
RS



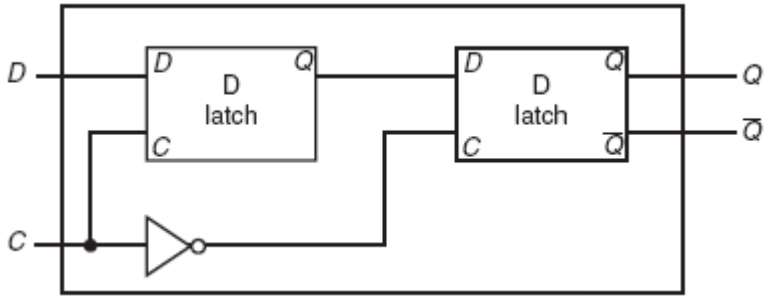


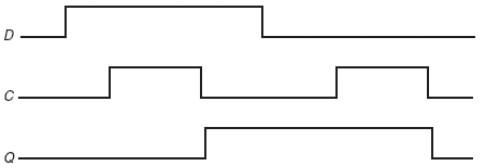
D latch, level-controlled flip-flop





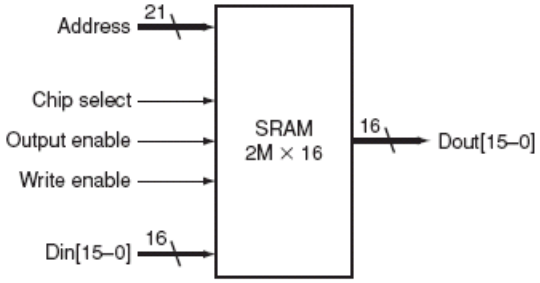
D flip-flop, edge-controlled flip-flop



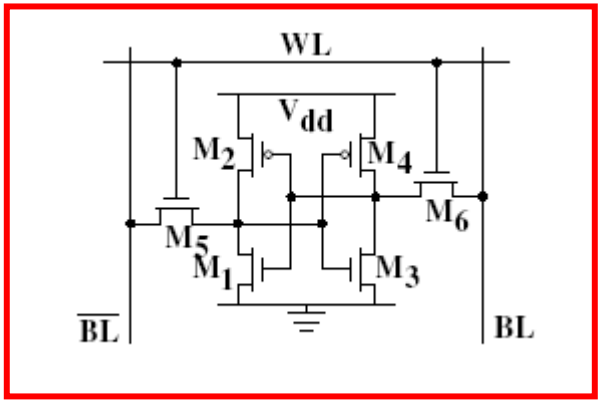


Usual SRAM Chip and SRAM Cell

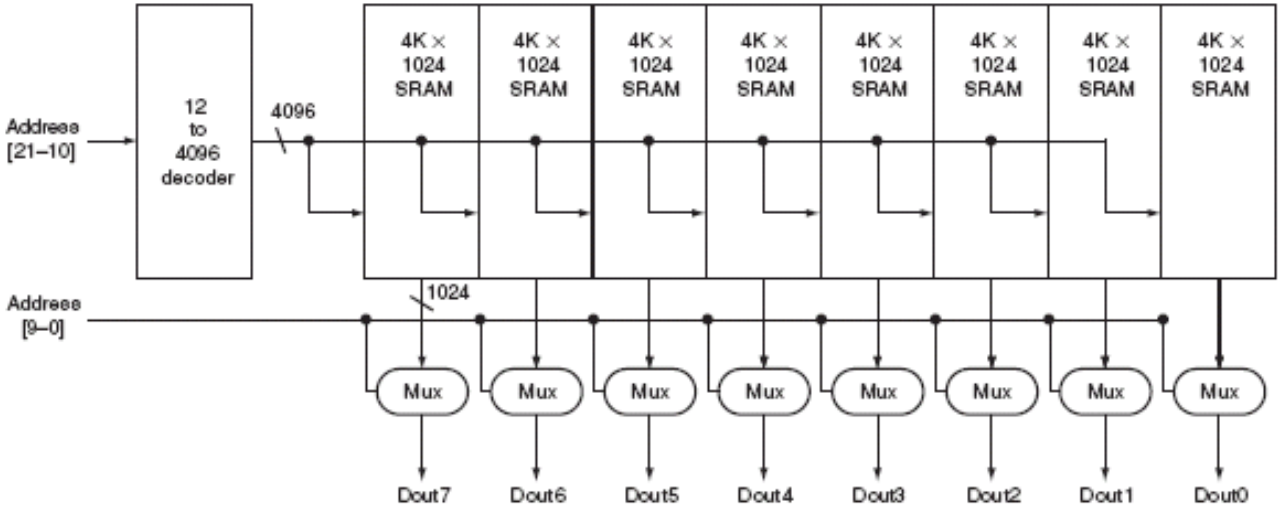
Usual SRAM chip



SRAM memory cell
CMOS technology

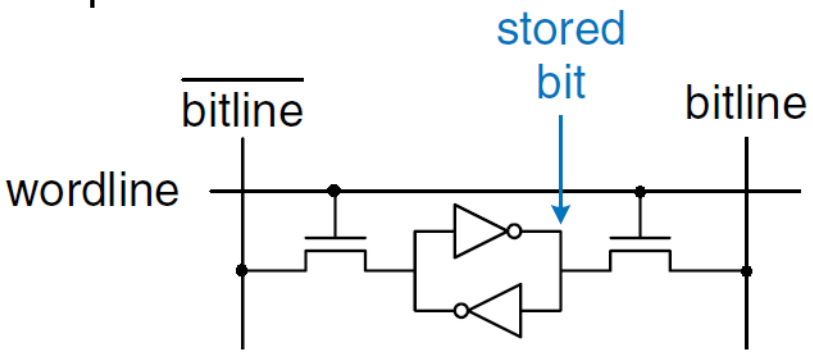


Bigger memory size?



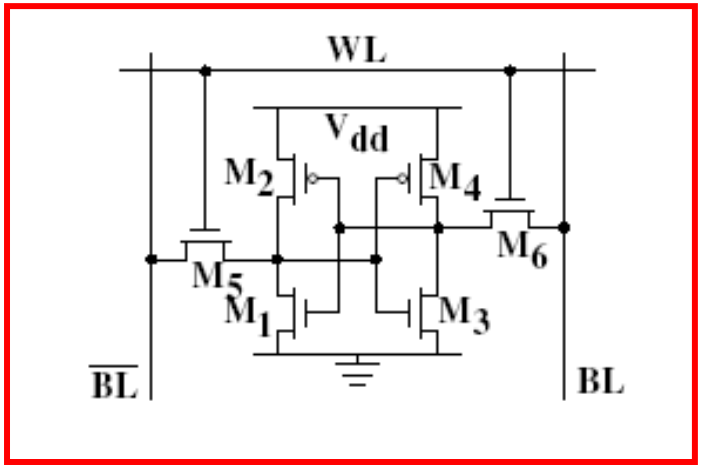
Usual Static Memory Chip Cell

Principle:

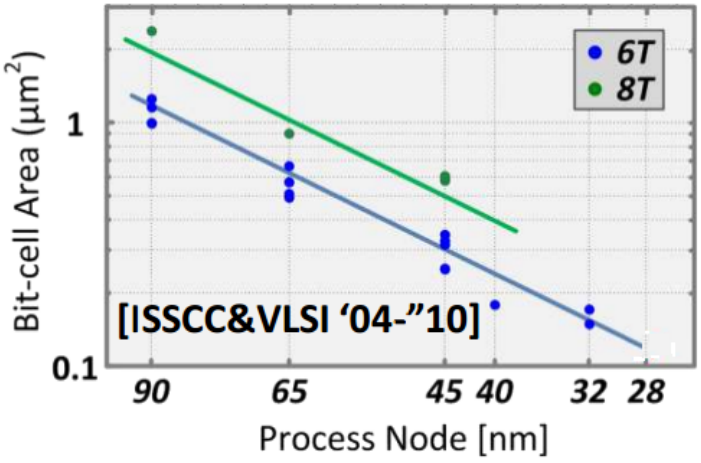


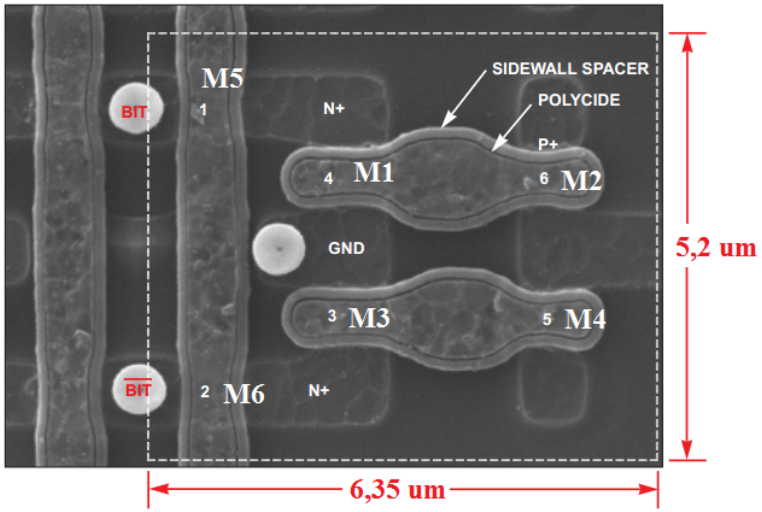
SRAM memory cell

6-transistors CMOS, 4 trans. Version exists



Area of one memory cell(bit):

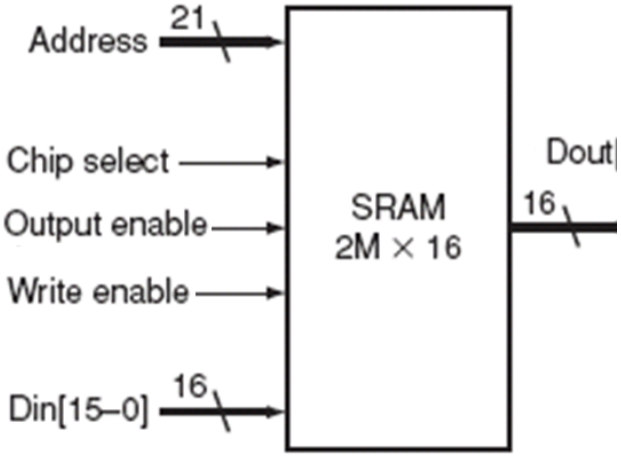




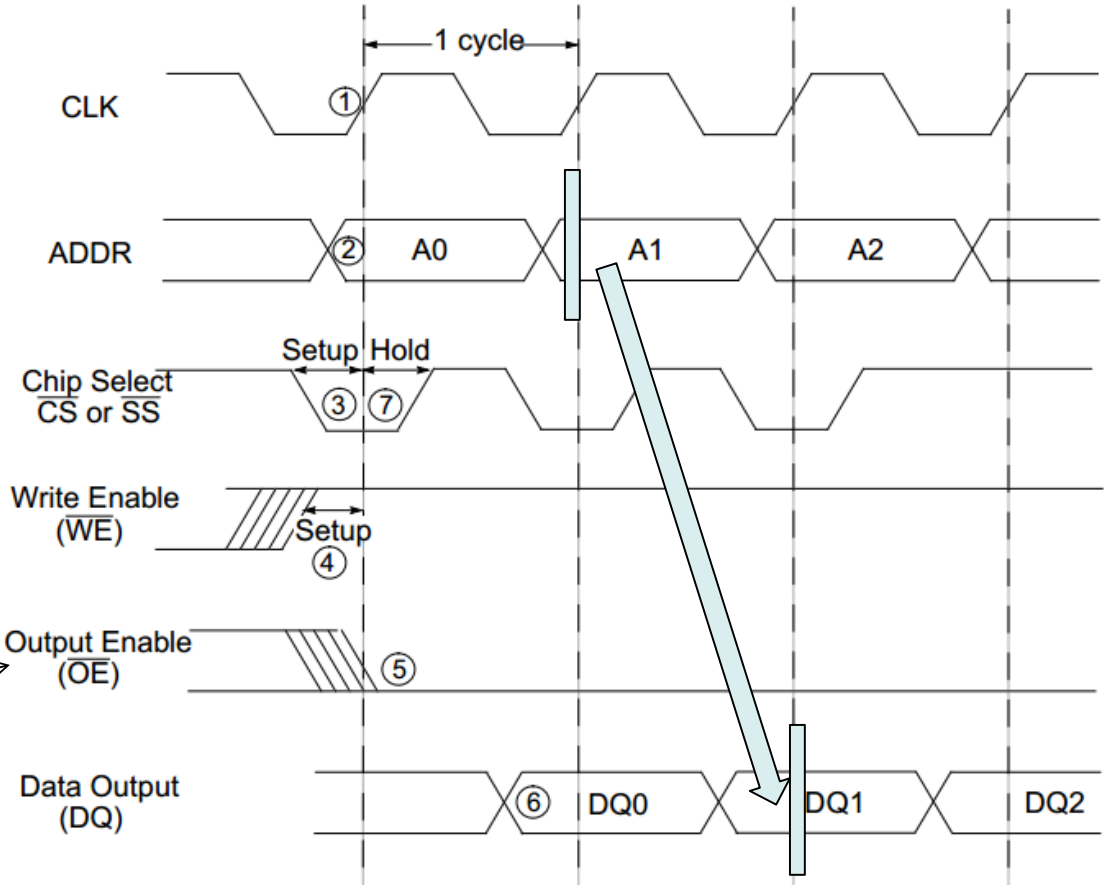
http://educypedia.karadimov.info/library/SEC08.PDF

Usual SRAM Chip

Typical synchronous SRAM chip



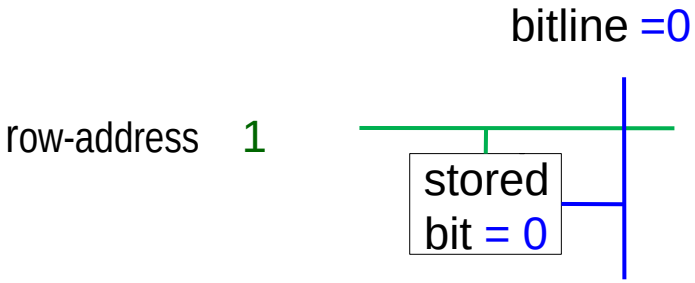
Read example for synchronous case:

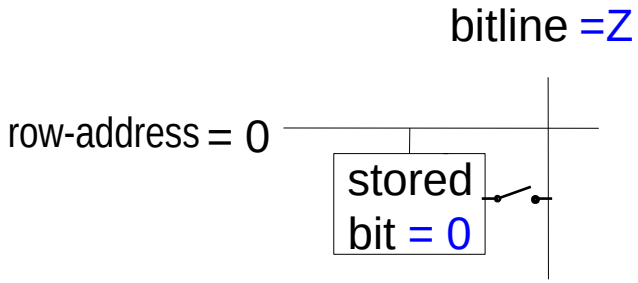


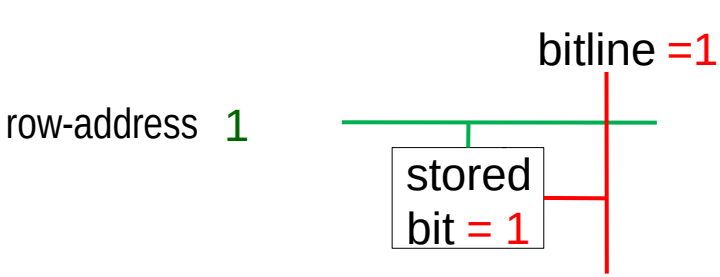
OE signal can be asynchronous

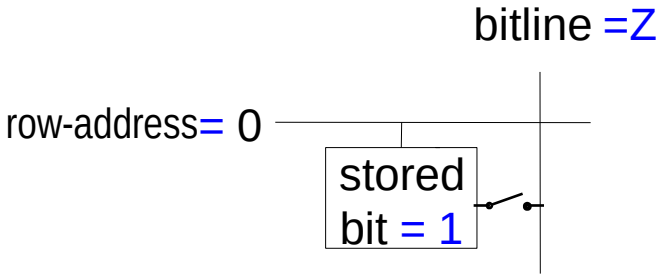
https://www.ece.cmu.edu/~ece548/localcpy/sramop.pdf

Memory Cell Connection to Matrix



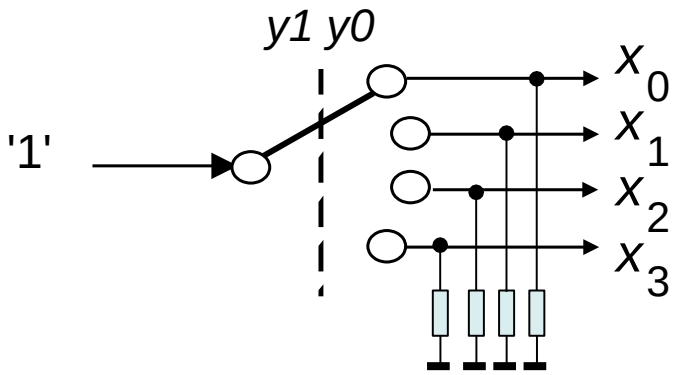


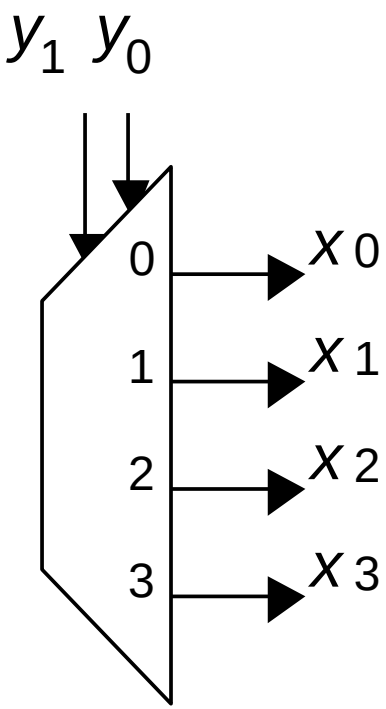




Selector Switch – One from N Decoder

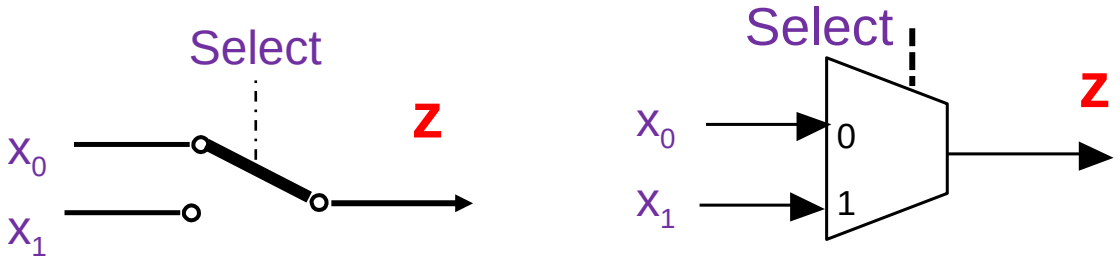
One Hot Decoder *cz: Dekodér 1 ze 4*



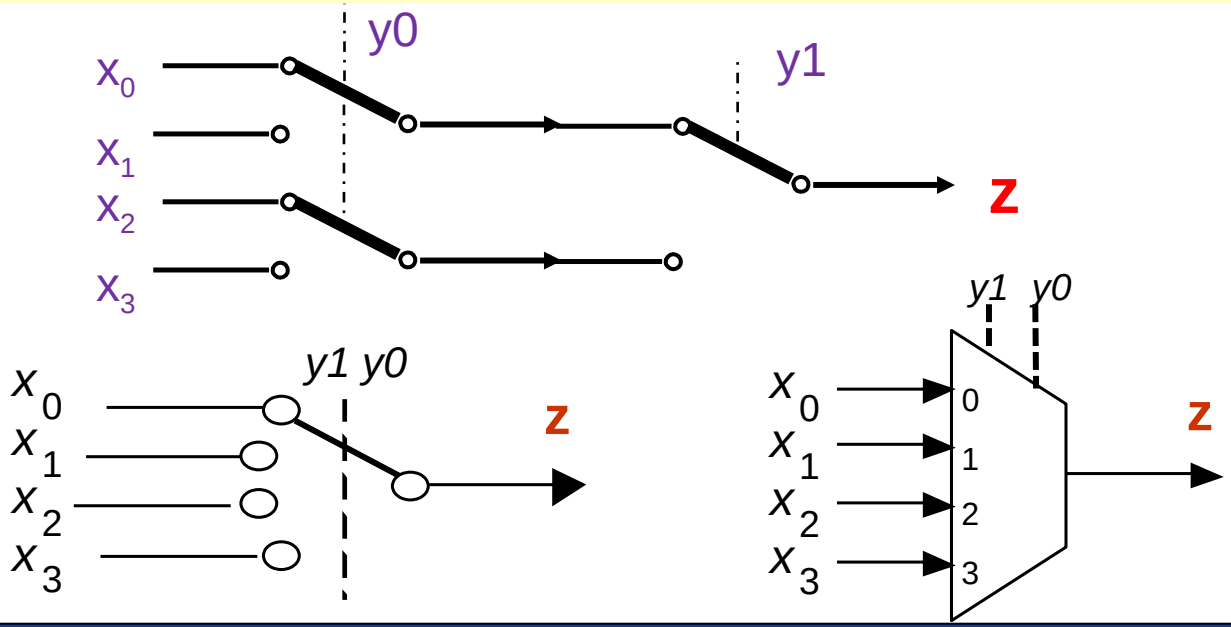


Switch Analogy of Multiplexer

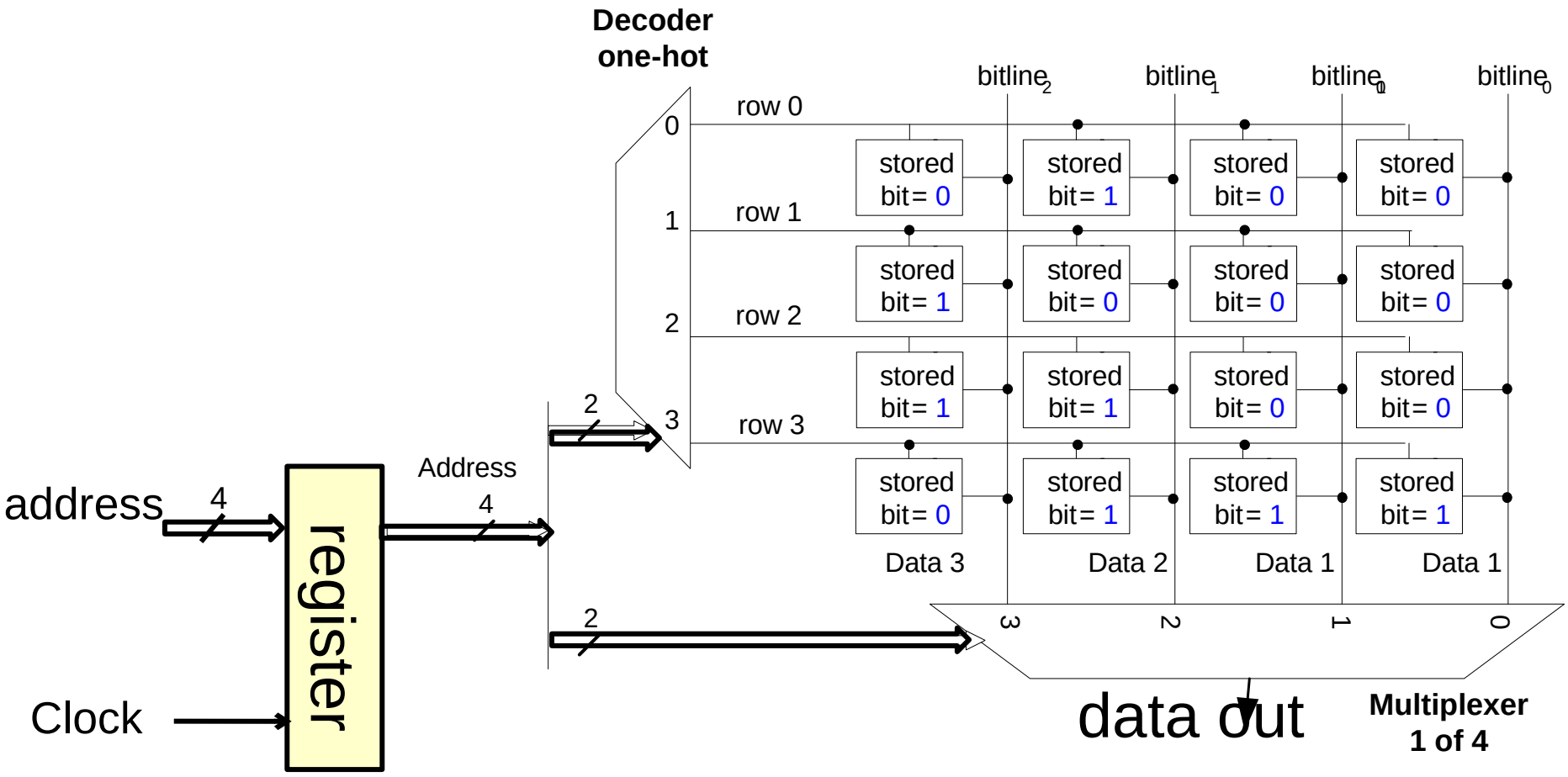
Multiplexer 2 to 1 or **1 of 2** *cz :2 kanálový (2-vstupový) multiplexor*



Multiplexer 4 to 1 or **1 of 4** *cz : 4 kanálový (4-vstupový) multiplexor*

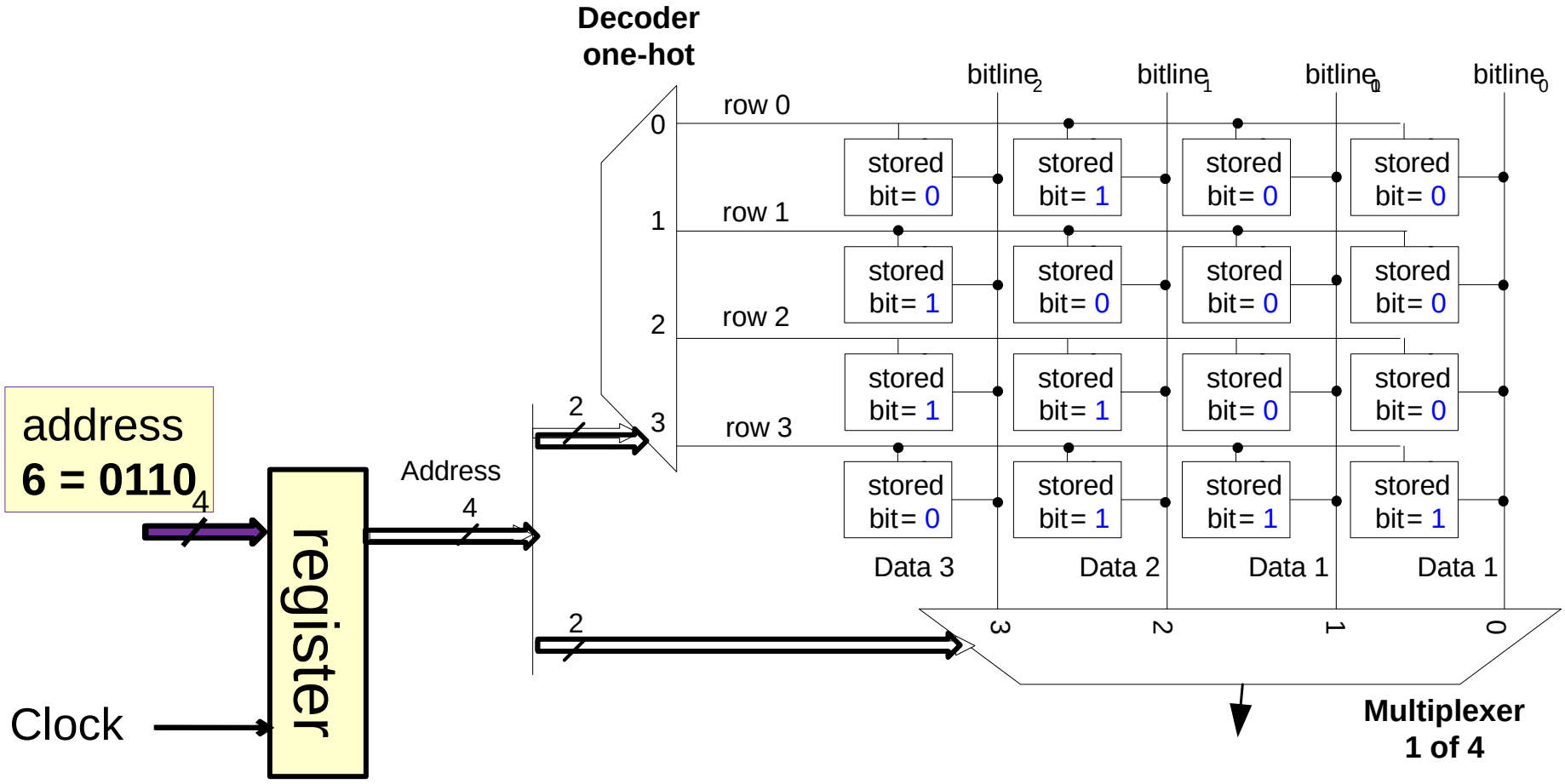


Memory Matrix



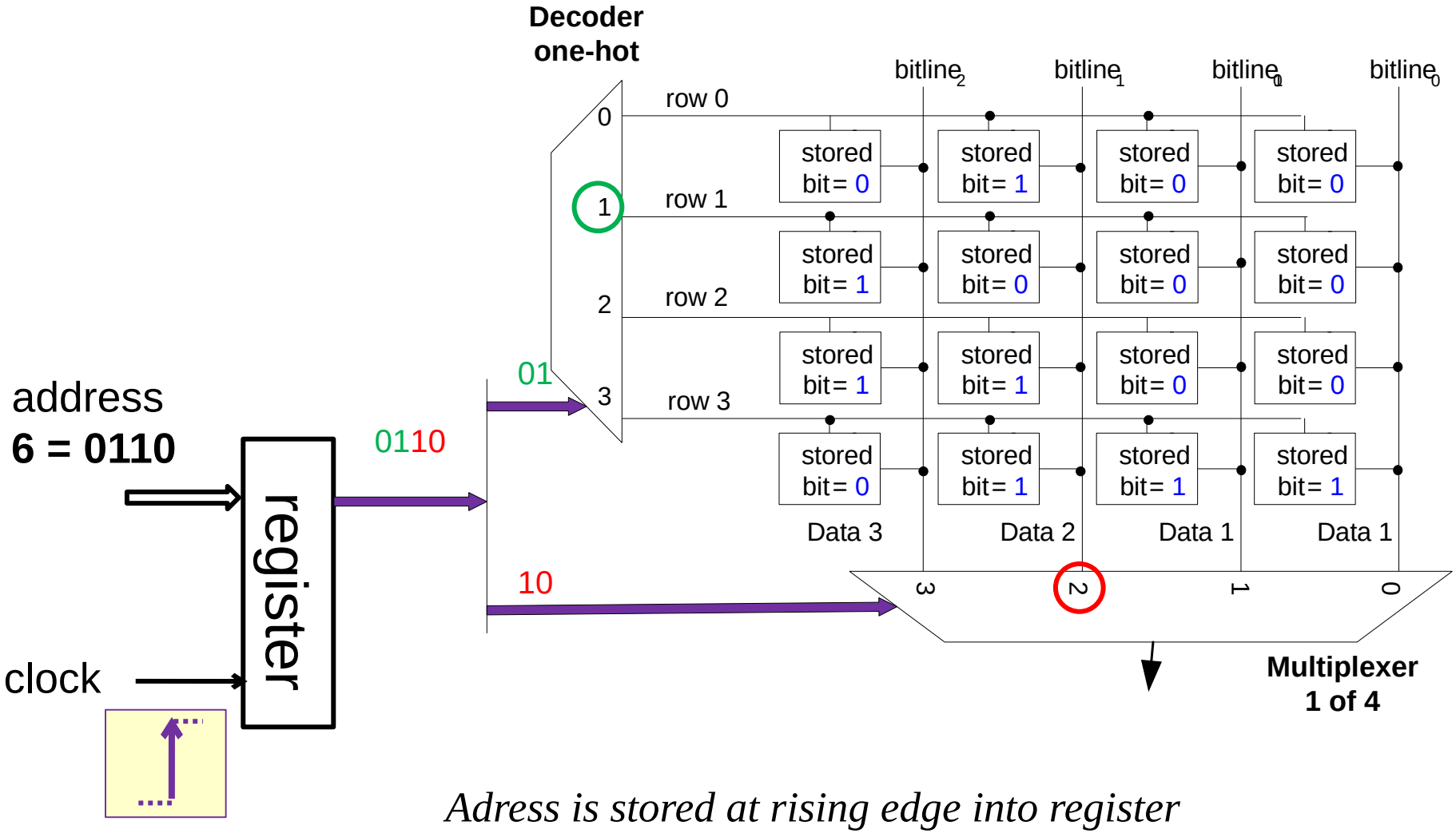
Register is necessary for synchronous memory implementation (SDRAM)

Memory Matrix – Operation



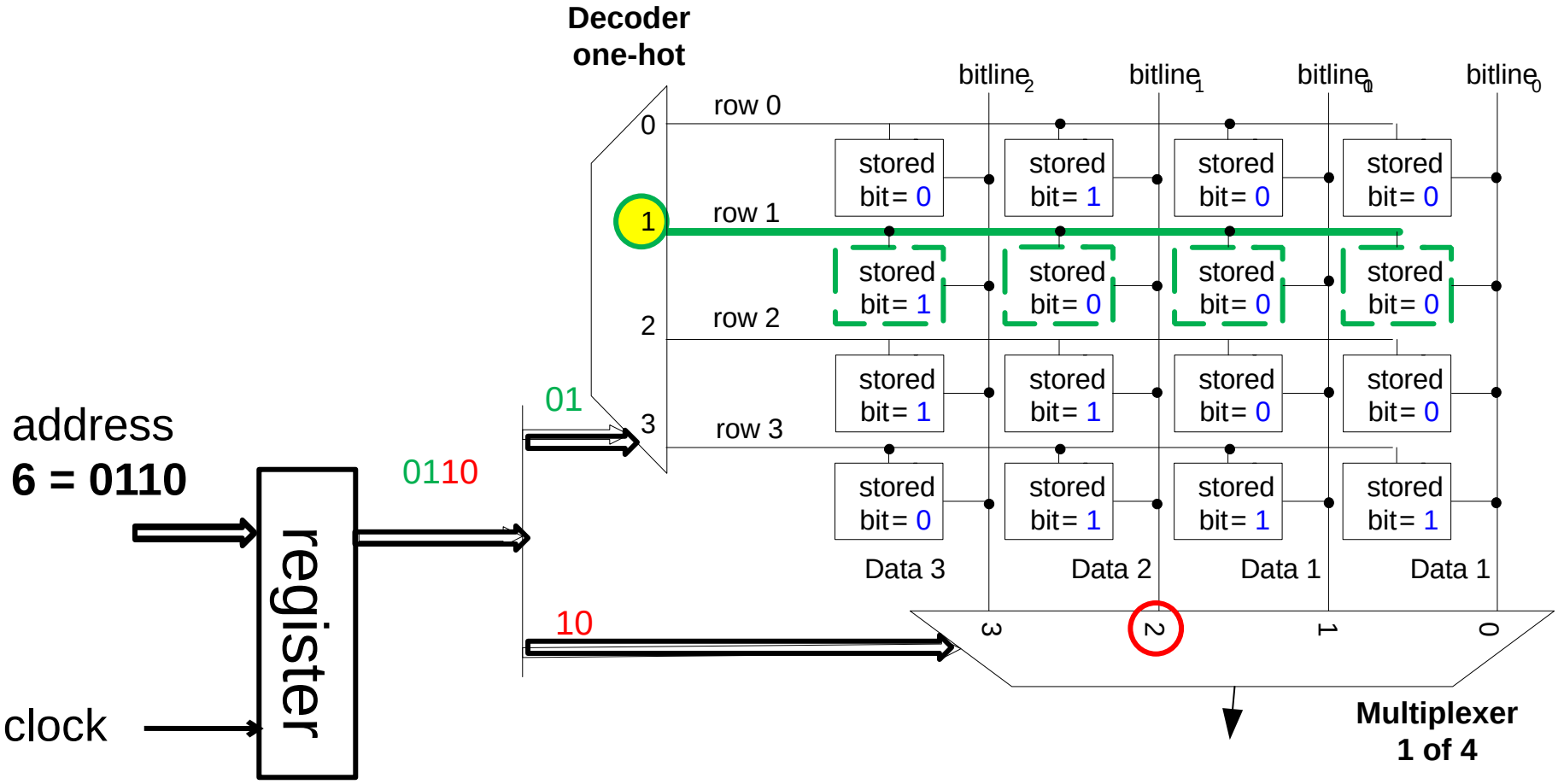
Address is setup at input and it is confirmed by rising edge.

Memory Matrix – Operation



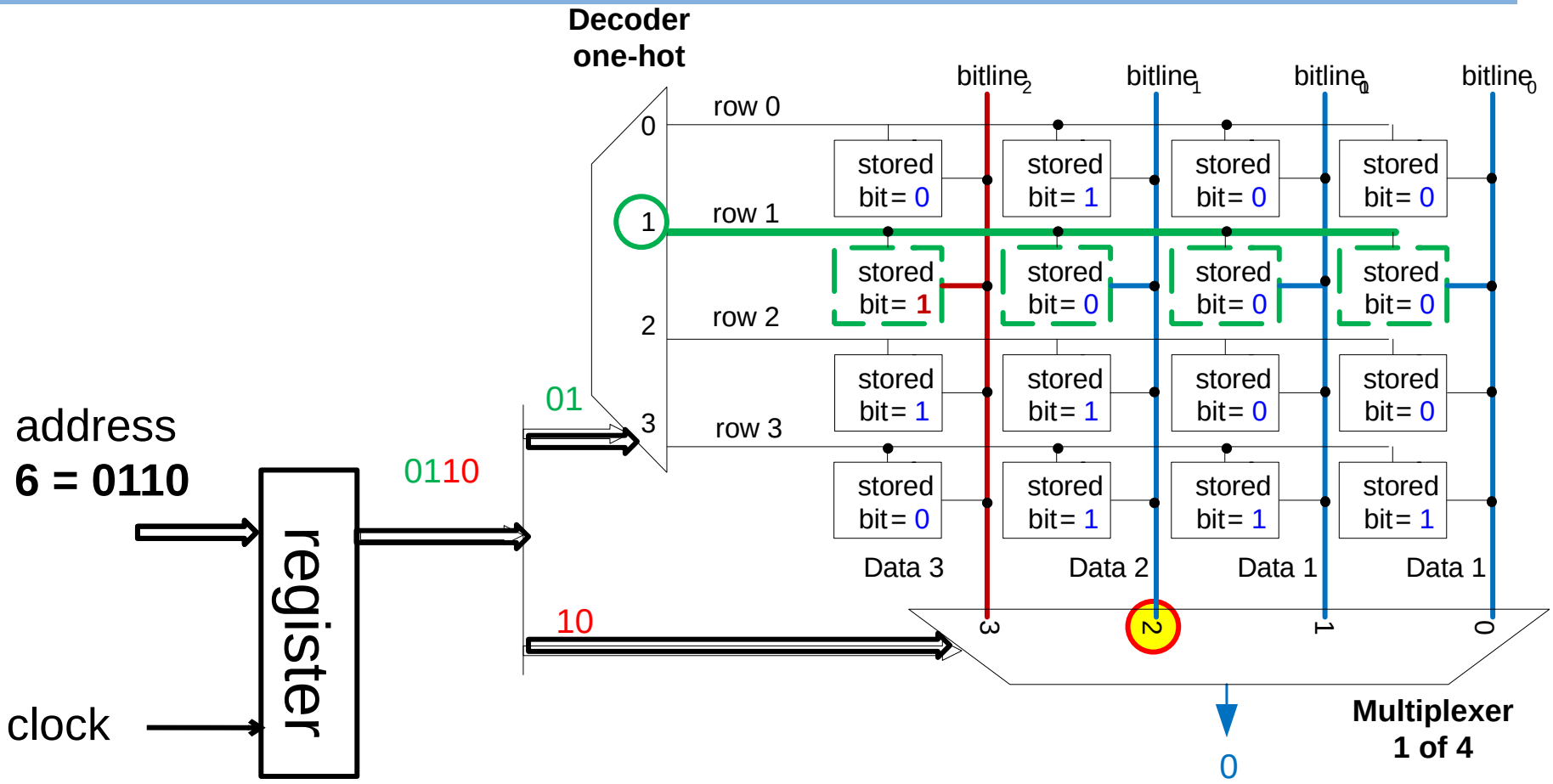
Adress is stored at rising edge into register and MSB bits select row and LSB bits column

Memory Matrix – Operation



Decoder activates 1 of N rows and the selected cells are connected to all columns bitlines

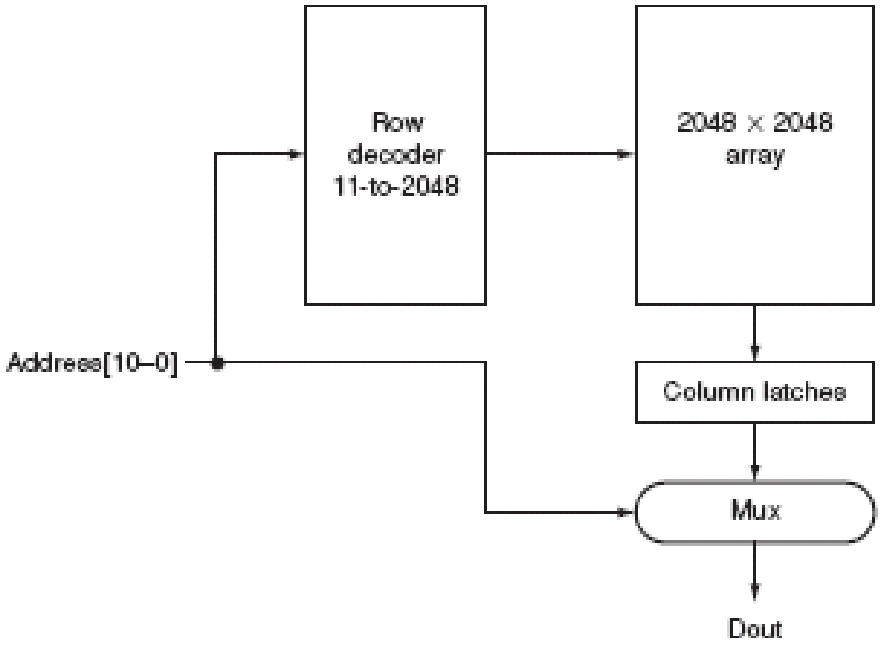
Memory Matrix – Operation

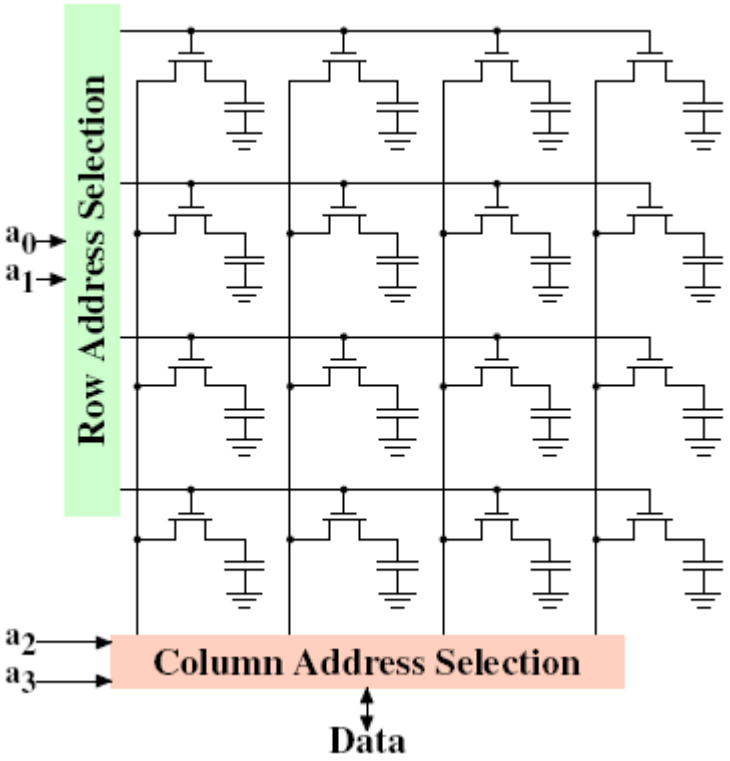


Multiplexer selects column - Data 2 = 0

When register is connected before multiplexer then whole row can be read at once and consecutive data words can be streamed out by multiplexer only switching columns

Internal Architecture of the DRAM Memory Chip

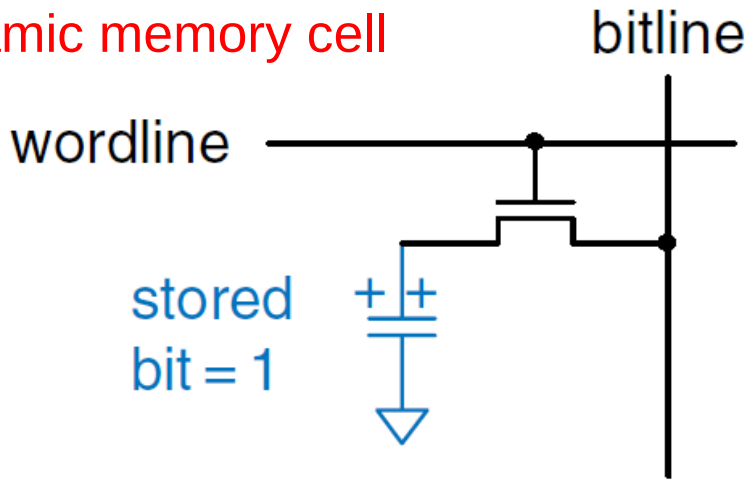


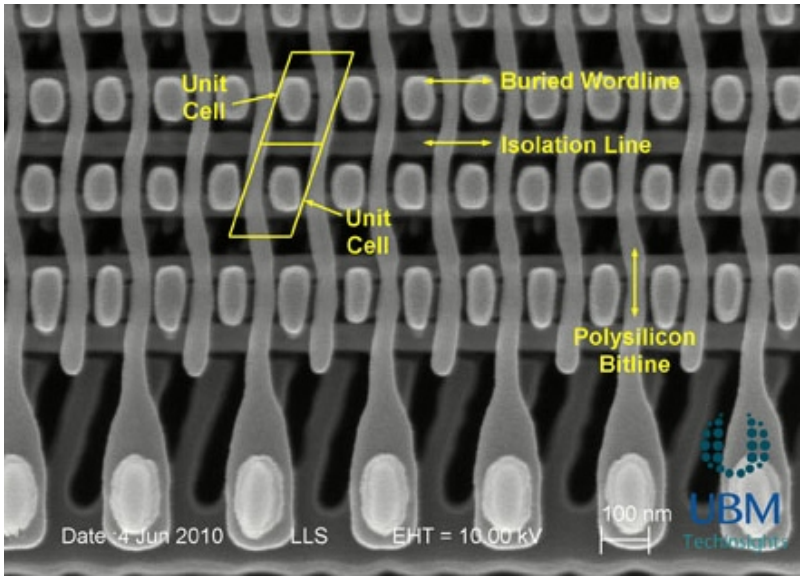


This 4M × 1 DRAM is internally realized as an 2048x2048 array of 1b memory cells

Detail of Dynamic Memory Cell

Single transistor
dynamic memory cell





- nMOS transistor nMOS works as analog switch which connects selected cell to „bitline“.
- „wordline“ controls which capacitor is connected to “bitline”

Source: http://www.eetimes.com/document.asp?doc_id=1281315

Dynamic Memory Capacitor Parameters

Today DRAM parameters	
	Capacity fF [femtofarad]
Capacitor capacity	from 10 fF to 50 fF
Bit line capacity	about 2 fF

[Source: l'INSA de Toulouse]

fF - femtofarad

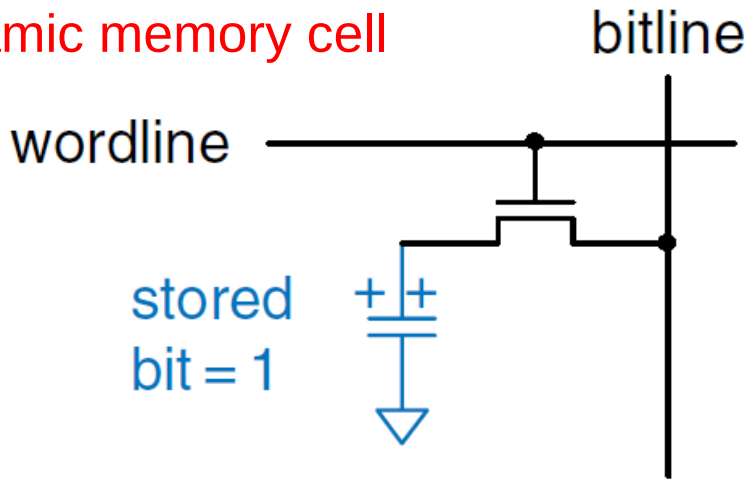
fF is SI unit equal to 10^{-15} Farads.

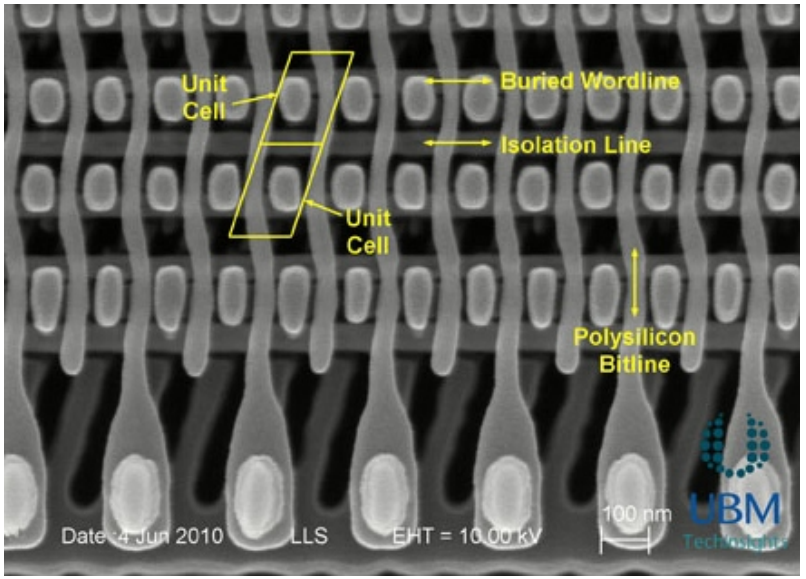
10^{-6} F = **1 µF** = 10^{3} nF = 10^{6} pF = 10^{9} fF

~9 fF is capacity between two plates of 1 mm^2 area with distance between plates around 1 mm,

Detail of Dynamic Memory Cell

Single transistor
dynamic memory cell



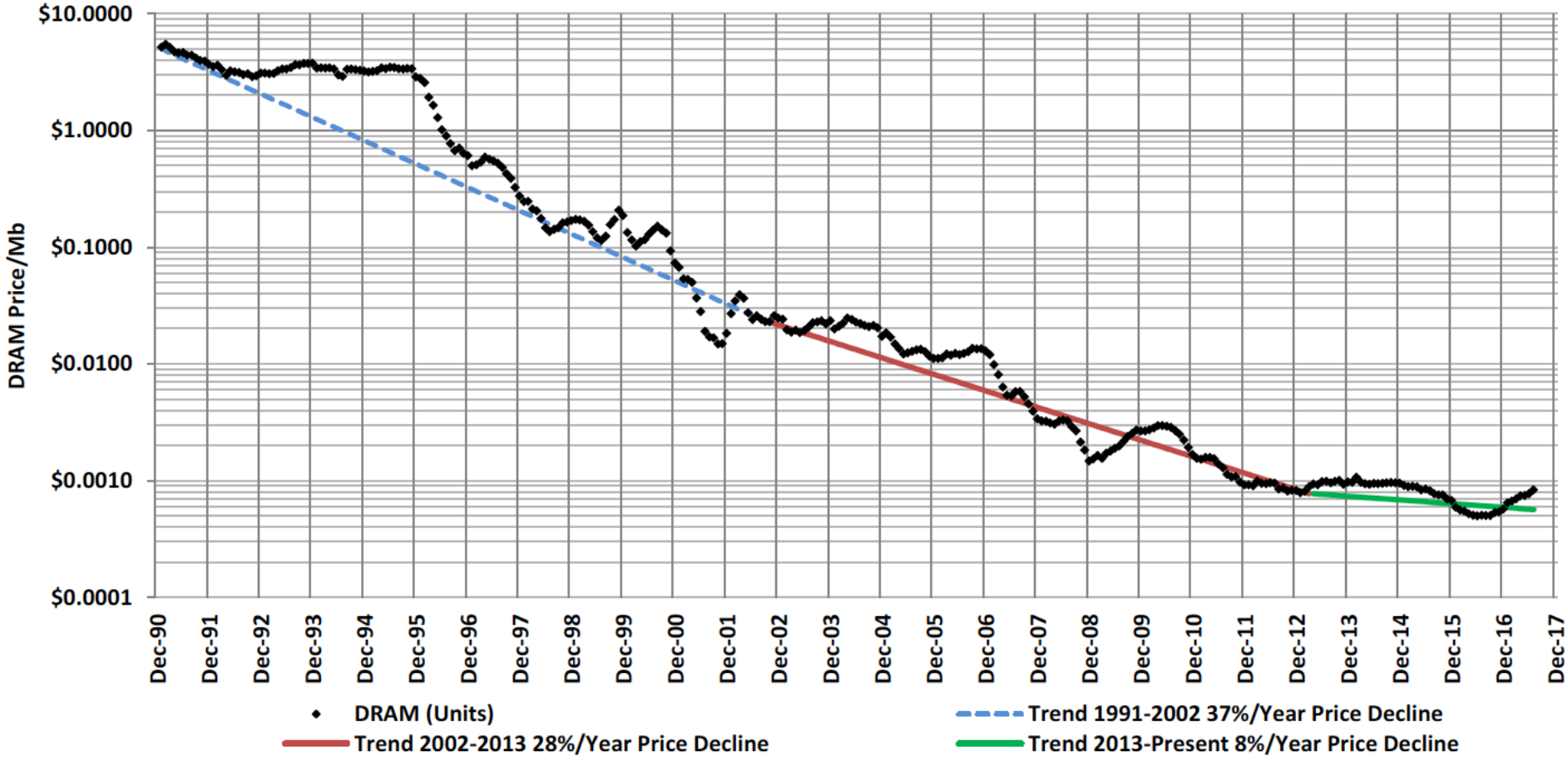


- Read operation is complex and slow, takes from 20 to 35 ns, and speedup is almost impossible
- Read is destructive, capacitor is discharged and original value has to be restored (refreshed) after each read.
- Femto-farad capacitor spontaneously discharges in short time
  - it is necessary to refresh it, in optimum case 60 ms for each cell, but maintenance frequency is multiplied by row count. Required refresh rate depends on temperature

Source: http://www.eetimes.com/document.asp?doc_id=1281315

DRAM Memories – Price Seems to Be Settled for Now

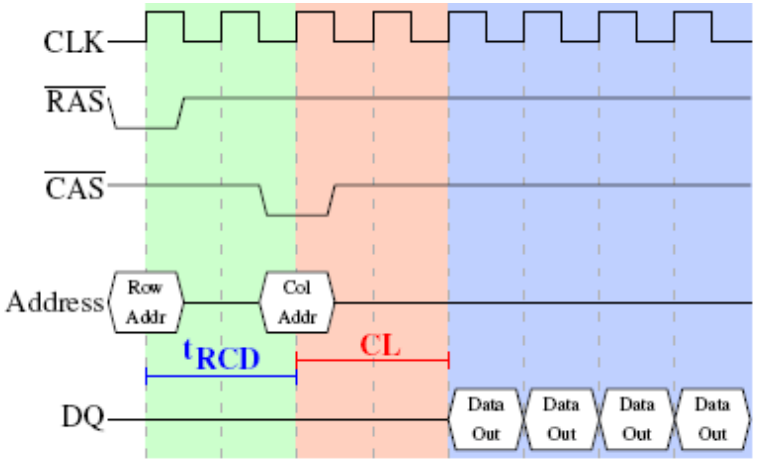
Price for 1 megabit



Source: Wells Fargo Securities, LLC and Semiconductor Industry Association

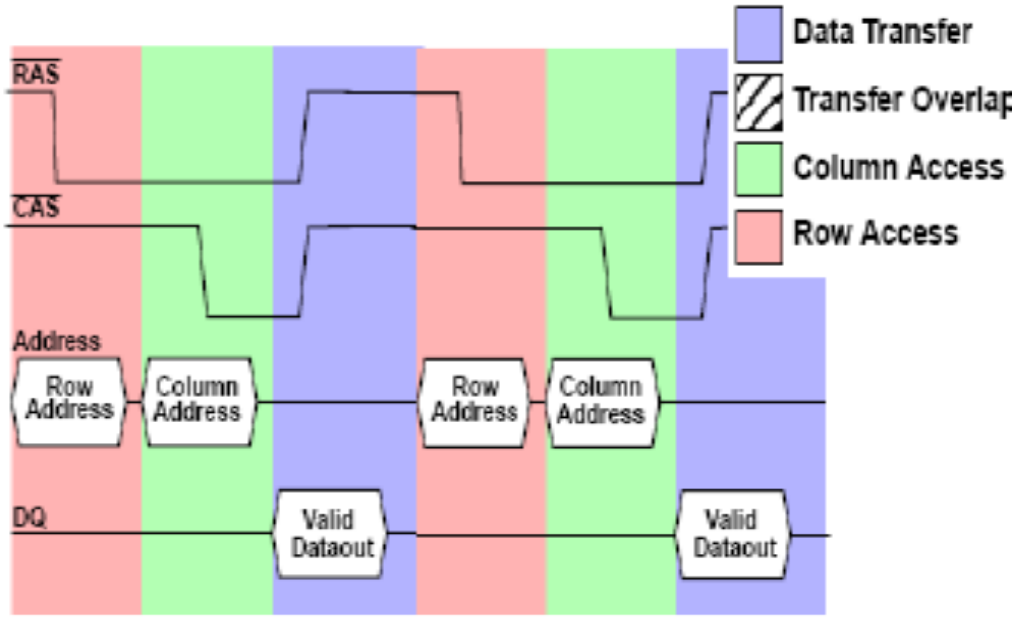
History of DRAM chips development

Year	Capacity	Price[$]/GB	Access time [ns]
1980	64 Kb	1 500 000	250
1983	256 Kb	500 000	185
1985	1 Mb	200 000	135
1989	4 Mb	50 000	110
1992	16 Mb	15 000	90
1996	64 Mb	10 000	60
1998	128 Mb	4 000	60
2000	256 Mb	1 000	55
2004	512 Mb	250	50
2007	1 Gb	50	40



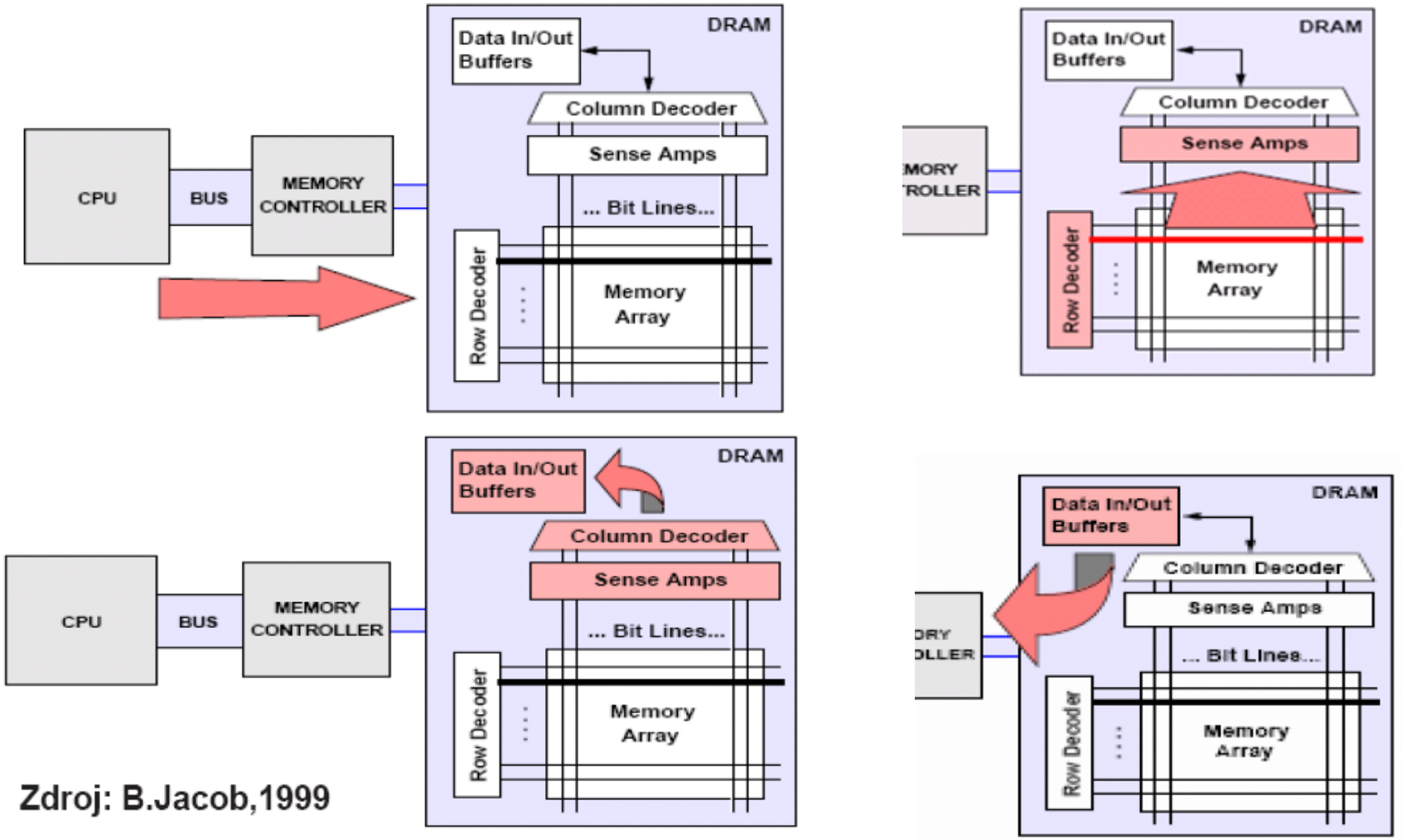
Old School DRAM – Asynchronous Access

- The address is transferred in two phases – reduces number of chip module pins and is natural for internal DRAM organization
- This method is preserved even for today chips



RAS – Row Address Strobe,
CAS – Column Address Strobe

Phases of DRAM Memory Read

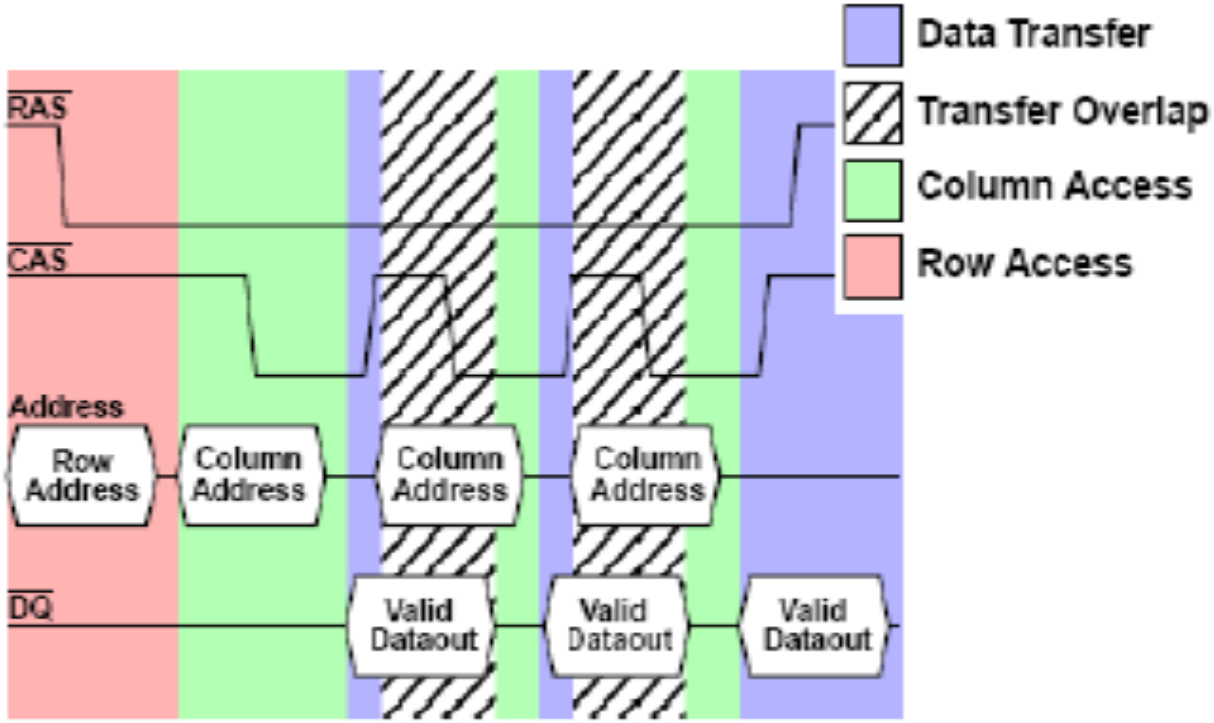


Zdroj: B.Jacob,1999

EDO-RAM – About 1995

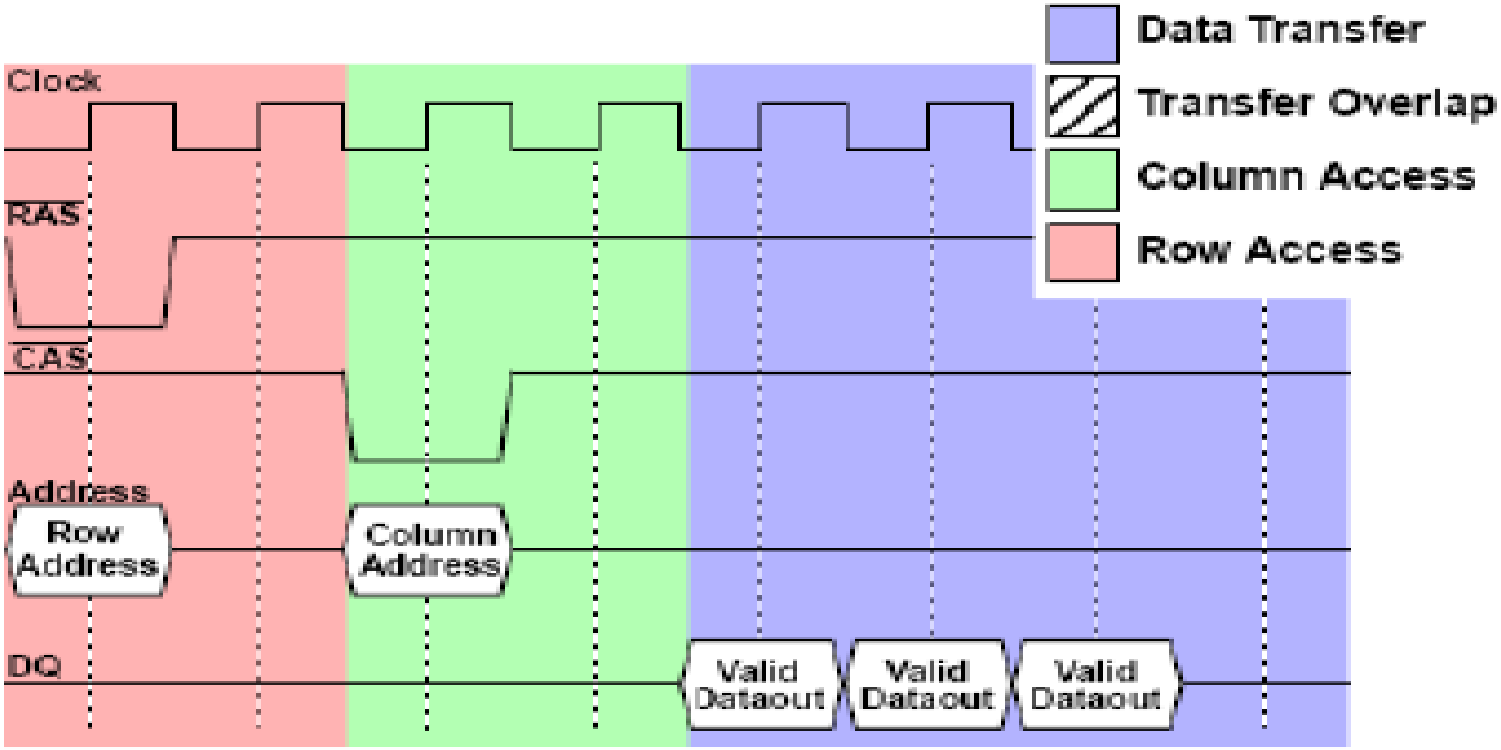
- Output register holds data during overlap of next read CAS phase with previous access data transfer

  this overlap (“pipelining”) increases throughput



SDRAM – end of 90-ties – synchronous DRAM

- SDRAM chip is equipped by counter that can be used to define continuous block length (burst) which is read together



SDRAM – the Most Widely Used Main Memory Technology

- SDRAM – clock frequency up to 100 MHz, 2.5V.
- DDR SDRAM – data transfer at both CLK edges, 2.5V, I/O bus clock 100-200 MHz, 0.2-0.4 GT/s (gigatransfers per second)
- DDR2 SDRAM – lower power consumption 1.8V, frequency up to 400 MHz, 0.8 GT/s
- DDR3 SDRAM – even lower power consumption at 1.5V, frequency up to 800 MHz, 1.6 GT/s
- DDR4 SDRAM – 1.05 – 1.2V, I/O bus clock 1.2 GHz, 2.4 GT/s
- DDR5 SDRAM – expected 2019-2020, ~6 GT/s
- All these innovations are focused mainly on throughput, not on the random access latency which for large capacities is still 20 to 35 ns.

Other Main Memory Types

- QDRx SDRAM (Quad Data Rate) – not twice as fast, allows only simultaneous read and write thanks to separated clocks for RD and WR, DDR are more effective than QDR for single access type only přístupu.
- GDDR SDRAM – today up to GDDR6, designed for graphics cards/GPUs

  - based on DDR memories.

  - data rate accelerated by wider output bus
- High Bandwidth Memory (HBM) is a high-performance RAM interface for 3D-stacked SDRAM from Samsung, AMD and SK Hynix.
- Another concept RDRAM (RAMBUS DRAM), which use completely different interface. Due to patent litigationare not in use in personal computers from 2003 year.

Notes for Today SDRAMs and Slides

- Use of the banked architecture that enables throughput to be increased by hiding latency of the opening and closing rows. These operations can proceed in parallel on different banks (sequential and interleaved banks mapping). The change result in a minimal pin count increase that is critical for price and density.

- Ulrich Drepper, Red Hat, Inc., What Every Programmer Should Know About Memory

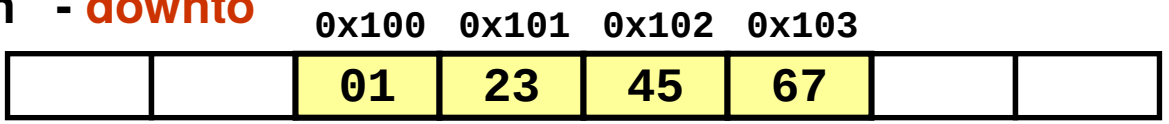
*Multi-byte Numbers

and their store in computer memory

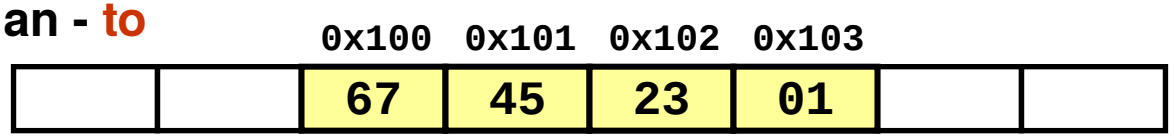
How to Store Multi-byte Number in Memory

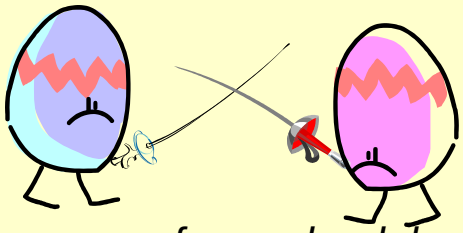
Hexadecimal number: 0x1234567

Big Endian - downto



Little Endian - to





Little-Endien comes from a book by Gulliver's Travels, Jonathan Swift 1726, in which he referred to one of the two opposing factions of the Lilliput. Ones ate eggs from the narrow end to the broader while
Big Endien proceeded the other way around. And the war did not wait long ...

Do you remember how war ended?

Memory Alignment (cz:zarovnání paměti?)

`.align n` directive

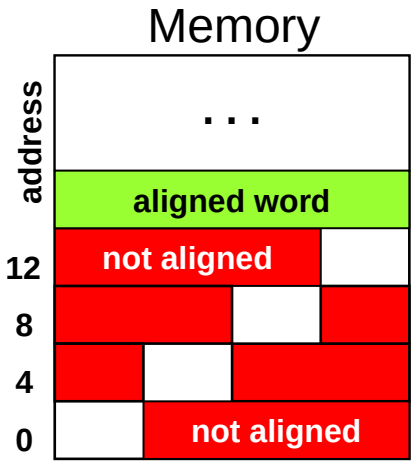
- next space allocated for data or text starts at 2^n divisible address

Example .align 2

- two least significant bits (LSB) are equal to **00**

Memory is addressed as byte array us usually (in C more precisely as array of **char**s)

The word of 32-bit processor is formed of 4-bytes in such case



Align in Data Segment Filled by Assembler

```
.data
.align  2  // or .align 4 on x86, use .p2align and .baling
    var1:  .byte   3, 5,'A','P','O'
.align  2  // or .align 4 on x86, use .p2align and .baling
    var2:  .word   0x12345678 // or .long on x86
.align  3  // or .align 8 on x86, use .p2align and .baling
    var3:  .2byte   1000 // or .word on x86
```

var1 var2

BIG ENDIAN	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x2000	3	5	41	50	4F				12	34	56	78				
0x2010	10	00														

var3 var1 var2

LITTLE ENDIAN	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x2000	3	5	41	50	4F				78	56	34	12				
0x2010	00	10														

var3

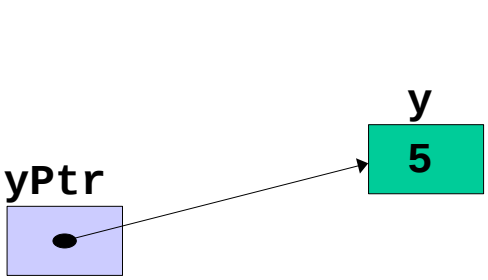
& (address operator)

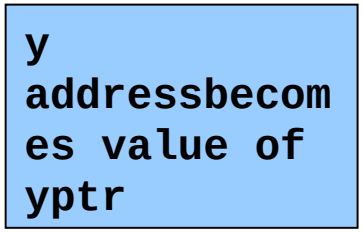
Returns the lowest address in memory address space where space/cells allocated to store variable starts.

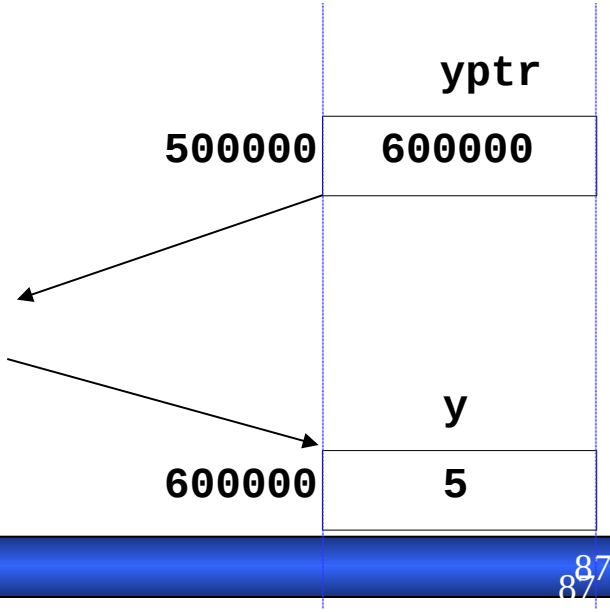
Example

```
int y = 5;
int *yPtr;
yPtr = &y;     // yPtr is signed to y address
```

yPtr “points to” **y**







C Language: Pointer Operations

& (address operator)

returns address of operand

* dereference address

returns value stored on address interpreted according to pointer type

***** and **&** are inverse
(but are not applicable in each case)

```
*&myVar == myVar
```

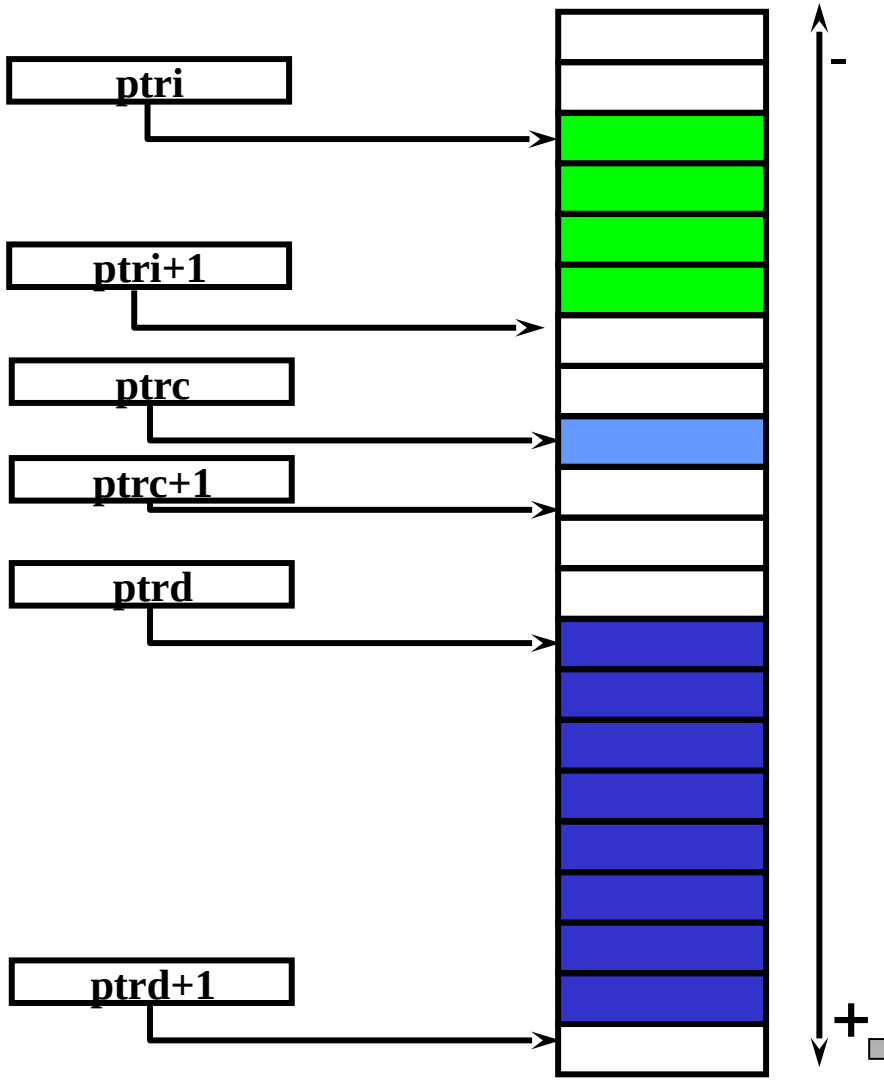
and

```
&*yPtr == yPtr
```

C Language: Size of Element Pointed by C Pointer

```
int * ptri;
char * ptrc;
double * ptrd;
```

*ptrx ≡ ptrx[0]
*(ptrx+1) ≡ ptrx[1]
*(ptrx+n) ≡ ptrx[n]
*(ptrx-n) ≡ ptrx[-n]



nr1 = sizeof (double);
nr2 = sizeof (double*);

nr1 != nr2

C Language: Pointer with const Qualifier

```
int x, y;
int * lpio = &y;
   *lpio = 1;x=*lpio;   lpio++;
const int * lpCio = &y;
   *lpCio = 1;  x=*lpCio;    lpCio++;
int * const lpioC = &y;
   *lpioC = 1; x=*lpioC;    lpioC++;
const int * const lpCioC = &y;
   *lpCioC = 1;   x=*lpCioC; lpCioC++;
```

C Language and Pointers

```
int i;
int *p;
p=&i;
```

```
i=i+1;
*p=*p+1;
i++;
(*p)++;
p[0]++;
```

```
int pole[30];
p=pole;
p=&pole[0];
```

```
for(i=0;i<30;i++)
  pole[i]++;
```

```
p=pole;
for(i=0;i<30;i++){
  (*(p++))++;
}
```

```
p++;
p=(int*) ((char*)p + sizeof(int));
```

The Lecture and Real Programming Question

Quick Quiz 1.: Is the result of both code fragments a same?

Quick Quiz 2.: Which of the code fragments is processed faster and why?

A:

```
int matrix[M][N];
int i, j, sum = 0;
…
for(i=0; i<M; i++)
  for(j=0; j<N; j++)
    sum += matrix[i][j];
```

B:

```
int matrix[M][N];
int i, j, sum = 0;
…
for(j=0; j<N; j++)
  for(i=0; i<M; i++)
    sum += matrix[i][j];
```

Is there a rule how to iterate over matrix element efficiently?