IMAGE PROCESSING I

Computer Technology

Michael Habeck

November 11, 2020

michael.habeck@uni-jena.de

Microscopic Image Analysis
University Hospital Jena

OVERVIEW

- An abridged history of computers
- Computer systems organization
- Logic gates and Boolean algebra
- Radix number systems
- Operators and data types in Python

AN ABRIDGED HISTORY OF COMPUTERS*

Year	Name	Made by	Comments
1623	Calculating clock	Schickard	Sketches of two mechanical calculators
1642	Pascaline	Pascal	Operated by rotating wheels
1673	Step reckoner	Leibniz	First true four-function calculator
1822	Difference engine	Babbage	Mechanical calculator for tabulating polynomials
1834	Analytical Engine	Babbage	First attempt to build a digital computer
1936	Z1	Zuse	First working relay calculating machine
1943	COLOSSUS	British gov't	First electronic computer
1944	Mark I	Aiken	First American general-purpose computer
1946	ENIAC	Eckert/Mauchley	Modern computer history starts here
1949	EDSAC	Wilkes	First stored-program computer
1951	Whirlwind I	M.I.T.	First real-time computer
1952	IAS	Von Neumann	Most current machines use this design
1960	PDP-1	DEC	First minicomputer (50 sold)
1961	1401	IBM	Enormously popular small business machine
1962	7094	IBM	Dominated scientific computing in the early 1960s
1963	B5000	Burroughs	First machine designed for a high-level language
1964	360	IBM	First product line designed as a family
1964	6600	CDC	First scientific supercomputer

from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

*See wikipedia entry on the history of computing hardware for more information

AN ABRIDGED HISTORY OF COMPUTERS*

Year	Name	Made by	Comments
1965	PDP-8	DEC	First mass-market minicomputer (50,000 sold)
1970	PDP-11	DEC	Dominated minicomputers in the 1970s
1974	8080	Intel	First general-purpose 8-bit computer on a chip
1974	CRAY-1	Cray	First vector supercomputer
1978	VAX	DEC	First 32-bit superminicomputer
1981	IBM PC	IBM	Started the modern personal computer era
1981	Osborne-1	Osborne	First portable computer
1983	Lisa	Apple	First personal computer with a GUI
1985	386	Intel	First 32-bit ancestor of the Pentium line
1985	MIPS	MIPS	First commercial RISC machine
1985	XC2064	Xilinx	First field-programmable gate array (FPGA)
1987	SPARC	Sun	First SPARC-based RISC workstation
1989	GridPad	Grid Systems	First commercial tablet computer
1990	RS6000	IBM	First superscalar machine
1992	Alpha	DEC	First 64-bit personal computer
1992	Simon	IBM	First smartphone
1993	Newton	Apple	First palmtop computer (PDA)
2001	POWER4	IBM	First dual-core chip multiprocessor

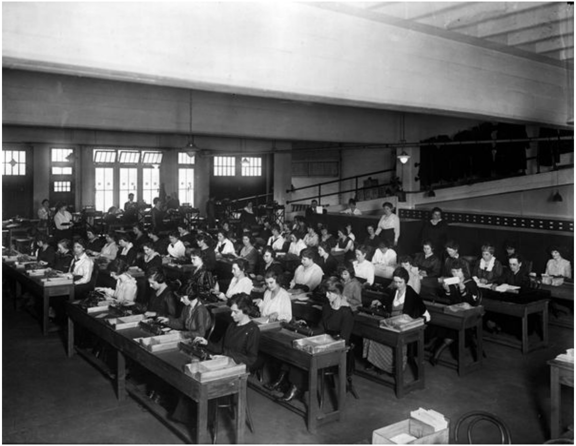
from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

*See wikipedia entry on the history of computing hardware for more information

AN ABRIDGED HISTORY OF COMPUTERS

from: www.computerhistory.org

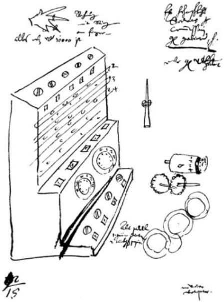
AN ABRIDGED HISTORY OF COMPUTERS



from: www.computerhistory.org

ZEROTH GENERATION—MECHANICAL COMPUTERS

Wilhem Schickard



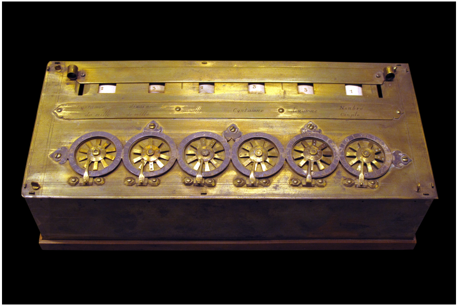
Sketch of his machine

Replica of Schickard's machine

from: https://commons.wikimedia.org

ZEROTH GENERATION—MECHANICAL COMPUTERS

Blaise Pascal

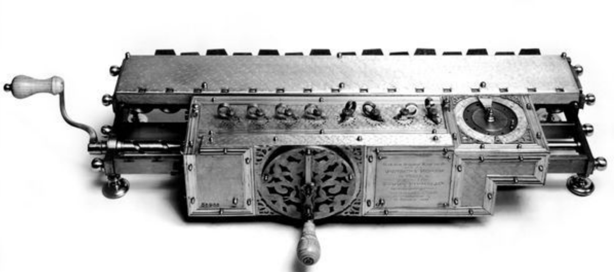


Replica of Pascal's machine, the *Pascaline*

from: https://commons.wikimedia.org

ZEROTH GENERATION—MECHANICAL COMPUTERS

Gottfried Leibniz



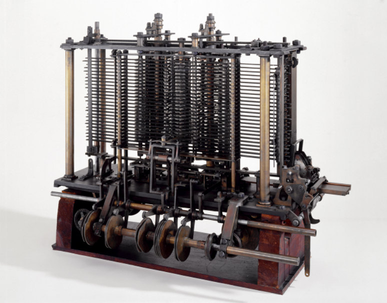
Replica of Leibniz's stepreckoner

… it is beneath the dignity of excellent men to waste their time in calculation when any peasant could do the work just as accurately with the aid of a machine.

from: www.computerhistory.org

ZEROTH GENERATION—MECHANICAL COMPUTERS

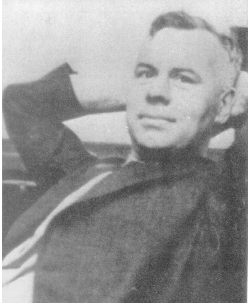
Charles Babbage



Replica of the analytical engine

from: wikipedia and www.computerhistory.org

FIRST GENERATION—RELAY BINARY ADDER

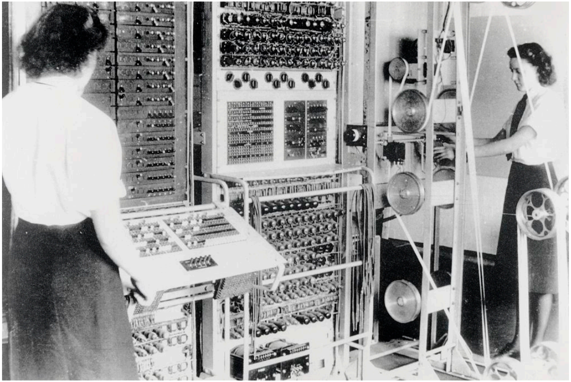


George Stibitz

Stibitz’s Model K binary adder

from: M. M. Irvine: IEEE Annals of the History of Computing (2001) 23:22–42

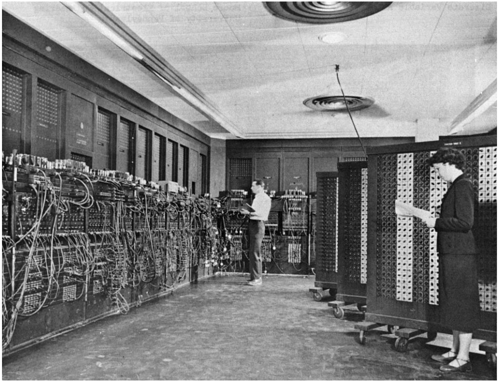
FIRST GENERATION—VACCUM TUBES



Colossus Mark 2

from: wikipedia

FIRST GENERATION—VACCUM TUBES

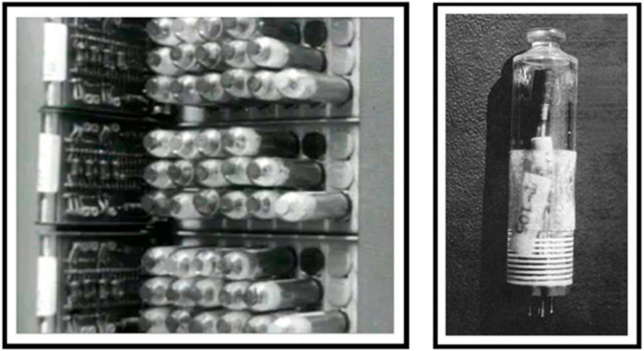


ENIAC (Electronic Numerical Integrator and Computer)

from: wikipedia

SECOND GENERATION—TRANSISTORS

MIT TX-0 Transistorized Computer Built in 1955, Operational in 1956



TX-0 (Transistorized eXperimental computer 0)

from: wikipedia

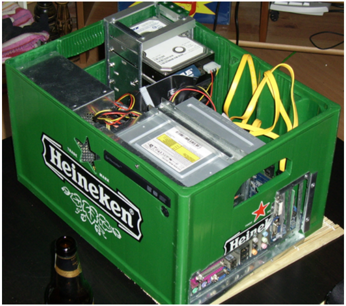
THIRD GENERATION—INTEGRATED CIRCUITS

IBM System/360

from: wikipedia

FOURTH GENERATION—VERY LARGE SCALE INTEGRATION

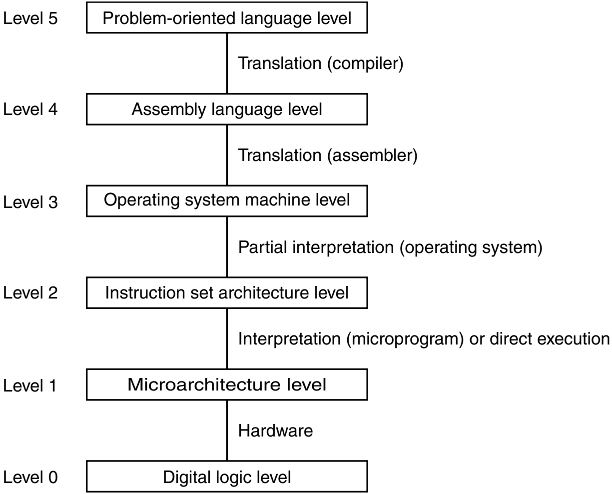
VLSI Chip



Today's computers

from: wikipedia and http://www.bytemods.com/mods/54/beer-case-mediacenter

CONTEMPORARY MULTILEVEL MACHINES



from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

Logic gates are physical implementations of **Boolean functions** with two inputs A, B and an output X[1]

Boolean functions can be represented by **truth tables**

Input A	Input B	Output X
0	0	X_1
0	1	X_2
1	0	X_3
1	1	X_4

1 represents **TRUE**, **ON**, etc.

0 represents **FALSE**, **OFF**, etc.

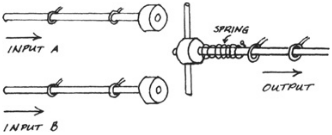
[1] General situation: n input *pins* and m output pins

LOGIC GATES AND BOOLEAN ALGEBRA

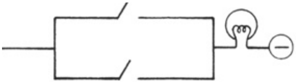
OR function

Input A	Input B	Output X
0	0	0
0	1	1
1	0	1
1	1	1

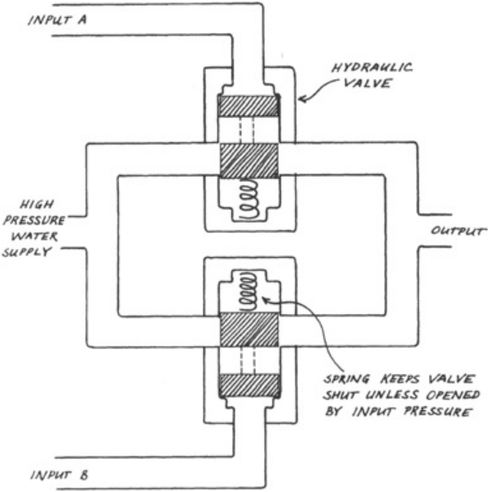
Mechanical OR gate



Electrical OR gate



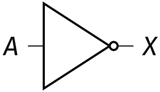
Hydraulic OR gate



from: W. D. Hillis: The Pattern on the Stone

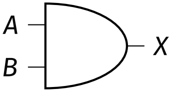
LOGIC GATES AND BOOLEAN ALGEBRA

NOT



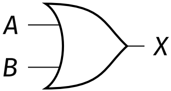
A	X
0	1
1	0

AND



A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

OR



A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

LOGIC GATES AND BOOLEAN ALGEBRA

Any logical function can be constructed from **NOT**, **OR**, and **AND** gates

A	B	C	D	$\cdots$	X		
0	0	0	0	$\cdots$	0		
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$		
0	0	0	1	$\cdots$	1	$\longrightarrow$	$\overline{A}\,\overline{B}\,\overline{C}\,D\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$		
0	1	1	0	$\cdots$	1	$\longrightarrow$	$\overline{A}\,B\,C\,\overline{D}\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$		

$$X = \cdots + \left(\overline{A}\,\overline{B}\,\overline{C}\,D\cdots\right) + \cdots + \left(\overline{A}\,B\,C\,\overline{D}\cdots\right) + \cdots$$

where

$$\overline{A} \leftrightarrow \mathrm{NOT}(A),\ A\,B \leftrightarrow \mathrm{AND}(A,B),\ A + B \leftrightarrow \mathrm{OR}(A,B)$$

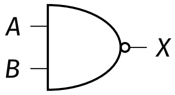
Another way to see that NOT, AND, OR suffice to represent any Boolean function is by recursion

$$
\begin{aligned}
X &= f(A, B, C, D, \ldots) \\
&= \overline{A}\, f(0, B, C, D, \ldots) + A\, f(1, B, C, D, \ldots) \\
&= \overline{A}\,\overline{B}\, f(0, 0, C, D, \ldots) + \overline{A}\, B\, f(0, 1, C, D, \ldots) + A\,\overline{B}\, f(1, 0, C, D, \ldots) + A\, B\, f(1, 1, C, D, \ldots) \\
&= \ldots
\end{aligned}
$$

NAND AND NOR

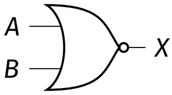
Can we simplify NOT, AND, OR any further?

NAND



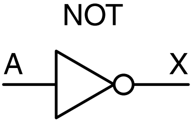
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

NOR

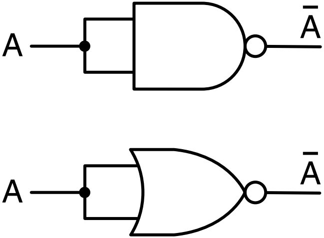


A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

CONSTRUCTING LOGICAL FUNCTIONS FROM NAND OR NOR

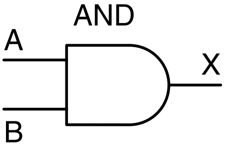


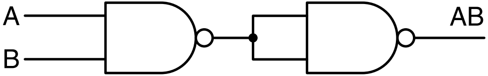
A	**X**
0	1
1	0



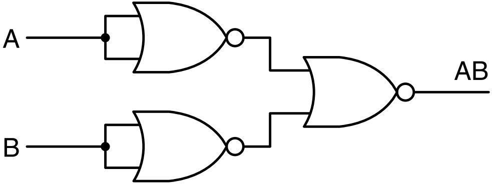
from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

CONSTRUCTING LOGICAL FUNCTIONS FROM NAND OR NOR



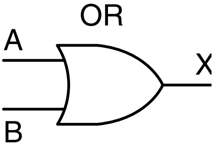


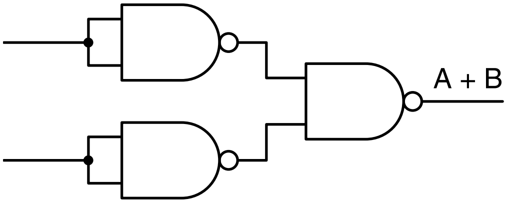
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1



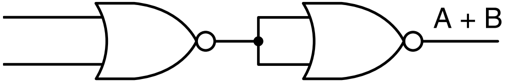
from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

CONSTRUCTING LOGICAL FUNCTIONS FROM NAND OR NOR



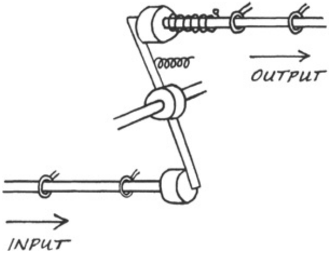


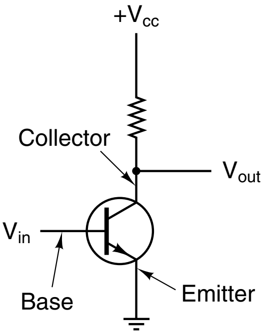
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1



from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

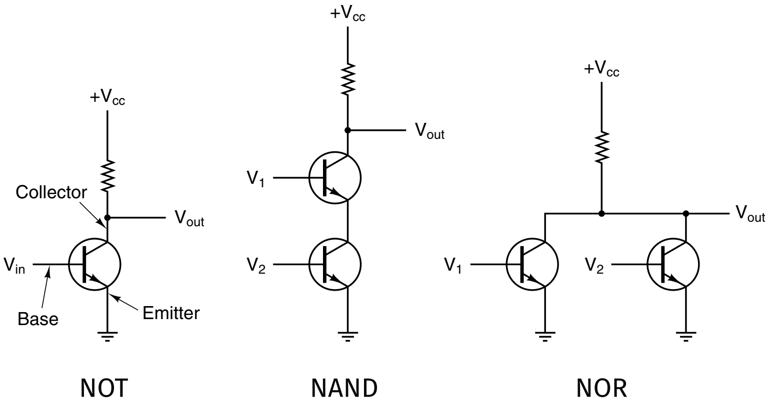
THE TRANSISTOR AS AN INVERTER





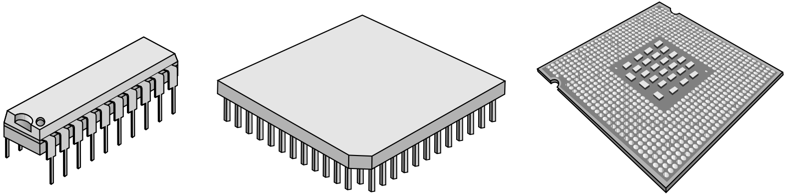
from: W. D. Hillis: The Pattern on the Stone
A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

CONSTRUCTING LOGICAL FUNCTIONS FROM NAND OR NOR



from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

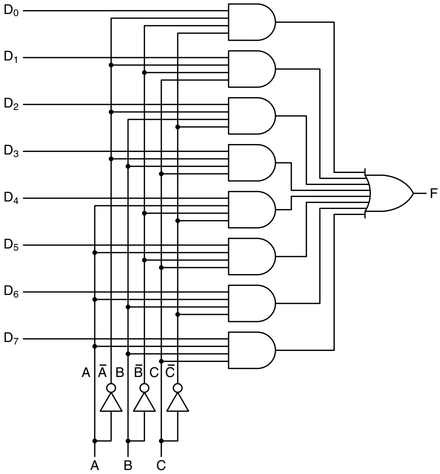
COMMON TYPES OF INTEGRATED CIRCUIT PACKAGES



from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

BASIC DIGITAL LOGIC CIRCUITS

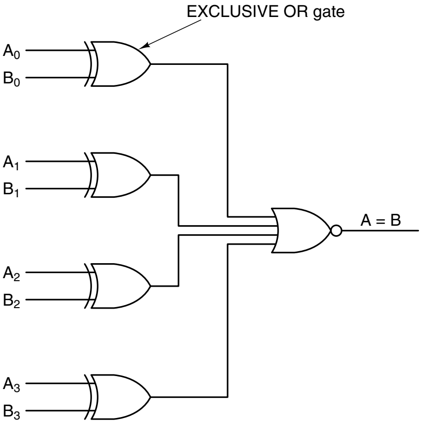
8-to-1 multiplexer



A	B	C	X
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

4-bit comparator



XOR

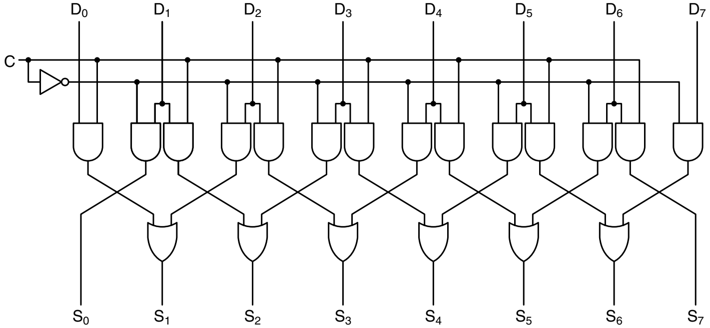
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

Left/right shifter: control bit C determines shifting direction

If C = 0 then $D_0D_1D_2D_3D_4D_5D_6D_7 \rightarrow D_1D_2D_3D_4D_5D_6D_7\,0$

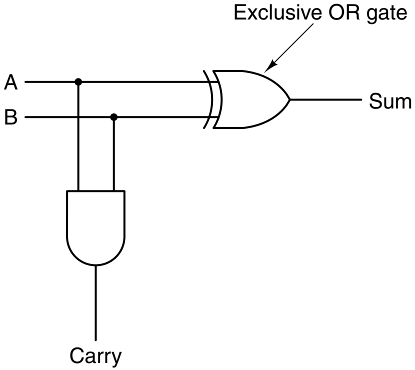
If C = 1 then $D_0D_1D_2D_3D_4D_5D_6D_7 \rightarrow 0\,D_0D_1D_2D_3D_4D_5D_6$



from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

BASIC DIGITAL LOGIC CIRCUITS

The half adder

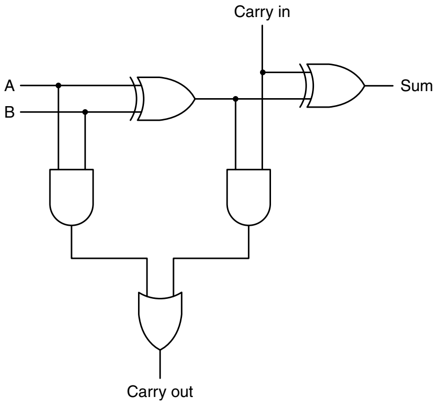


A	B	X	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

BASIC DIGITAL LOGIC CIRCUITS

The full adder

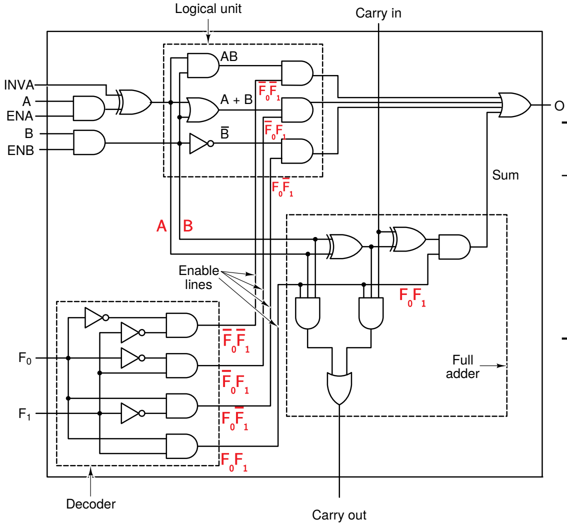


A	B	Carry	X	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

BASIC DIGITAL LOGIC CIRCUITS

1-bit arithmetic logic unit (ALU)

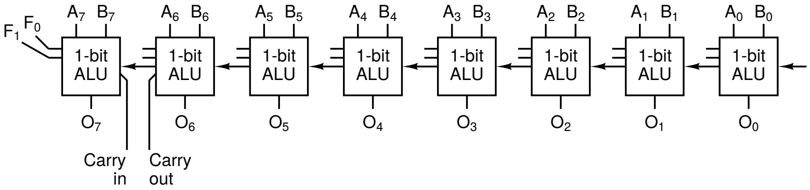


F_0	F_1	Function
0	0	AND(A, B)
0	1	OR(A, B)
1	0	NOT(B)
1	1	ADD(A, B)

from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

BASIC DIGITAL LOGIC CIRCUITS

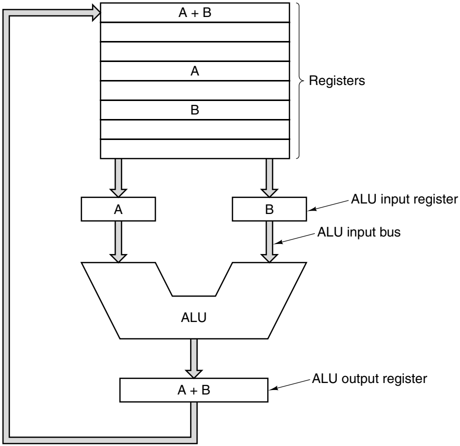
Eight 1-bit ALUs can be connected to form an 8-bit ALU



from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

BASIC DIGITAL LOGIC CIRCUITS

The data path of a typical von Neumann machine



from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

THE DECIMAL NUMBER SYSTEM

A decimal number has the following form:

		100's place		10's place	1's place		0.1's place	0.01's place	0.001's place		
		↓		↓	↓		↓	↓	↓		
d_n	···	d_2		d_1	d_0	.	d_{-1}	d_{-2}	d_{-3}	···	d_{-k}

In compact notation

$$\sum_{i=-k}^{n} d_i \times 10^i, \quad d_i \in \{0, 1, 2, \dots, 9\}$$

For a general base ***b***

$$\sum_{i=-k}^{n} d_i \times b^i, \quad d_i \in \{0, 1, 2, \dots, b-1\}$$

RADIX NUMBER SYSTEMS

The decimal number 2001 represented as binary, octal, and hexadecimal number

Binary

1	1	1	1	1	0	1	0	0	0	1
1×2^{10}	$+ 1 \times 2^9$	$+ 1 \times 2^8$	$+ 1 \times 2^7$	$+ 1 \times 2^6$	$+ 0 \times 2^5$	$+ 1 \times 2^4$	$+ 0 \times 2^3$	$+ 0 \times 2^2$	$+ 0 \times 2^1$	$+ 1 \times 2^0$
1024	+512	+256	+128	+64	+0	+16	+0	+0	+0	+1

Octal

3	7	2	1
3×8^3	$+ 7 \times 8^2$	$+ 2 \times 8^1$	$+ 1 \times 8^0$
1536	+448	+16	+1

Decimal

2	0	0	1
2×10^3	$+ 0 \times 10^2$	$+ 0 \times 10^1$	$+ 1 \times 10^0$
2000	+0	+0	+1

Hexadecimal

7	D	1
7×16^2	$+13 \times 16^1$	$+1 \times 16^0$
1792	+208	+1

CONVERSION BETWEEN NUMBER SYSTEMS

Decimal	Binary	Octal	Hex
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D

Decimal	Binary	Octal	Hex
14	1110	16	E
15	1111	17	F
16	10000	20	10
20	10100	24	14
30	11110	36	1E
40	101000	50	28
50	110010	62	32
60	111100	74	3C
70	1000110	106	46
80	1010000	120	50
90	1011010	132	5A
100	1100100	144	64
1000	1111101000	1750	3E8
2989	101110101101	5655	BAD

CONVERSION BETWEEN NUMBER SYSTEMS

Example 1

Hexadecimal	1 9 4 8 . B 6
Binary	0 0 0 1 1 0 0 1 0 1 0 0 1 0 0 0 . 1 0 1 1 0 1 1 0 0
Octal	1 4 5 1 0 . 5 5 4

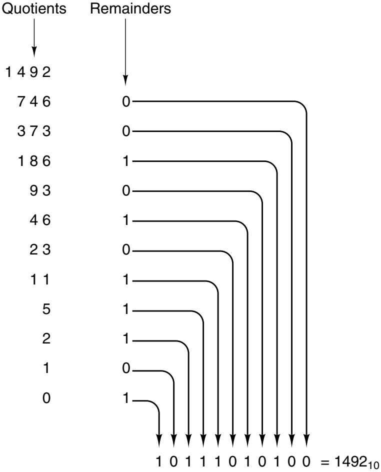
Example 2

Hexadecimal	7 B A 3 . B C 4
Binary	0 1 1 1 1 0 1 1 1 0 1 0 0 0 1 1 . 1 0 1 1 1 1 0 0 0 1 0 0
Octal	7 5 6 4 3 . 5 7 0 4

from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

BINARY NUMBERS

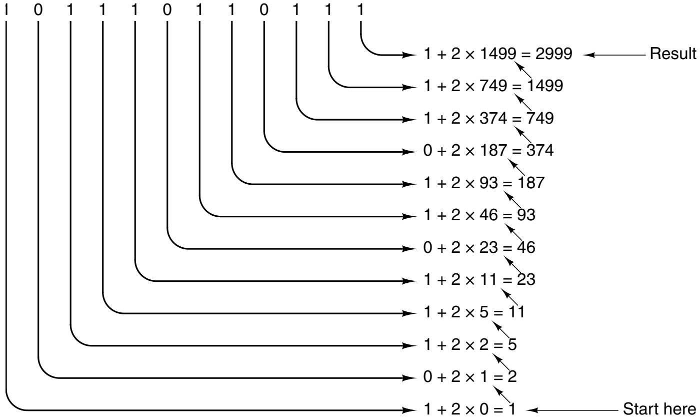
Conversion of decimal numbers to binary numbers by successive halving



from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

BINARY NUMBERS

Conversion of binary numbers to decimal numbers by successive doubling



from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

Binary arithmetic addition (compare with half adder)

Addend	0	0	1	1
Augend	+0	+1	+0	+1
Sum	0	1	1	0
Carry	0	0	0	1

A	*B*	*X*	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Example: adding two 8-bit binary numbers

Carry			*1*	*1*	*1*		*1*	*1*	*1*	
$(87)_{10}$			0	1	0	1	0	1	1	1
$(183)_{10}$	+		1	0	1	1	0	1	1	1
$(270)_{10}$		1	0	0	0	0	1	1	1	0

BINARY NUMBERS

To represent **negative binary numbers**, one bit (the *sign bit*) indicates the sign: $0/1 \leftrightarrow +/-$

Negative binary numbers can be represented in multiple ways

signed magnitude: leftmost bit is the sign bit

one's complement: complement of the magnitude of the number, i.e. the sum of an n-bit number and its one's complement is $2^n - 1$ (→wikipedia); numbers range from $\pm(2^{n-1} - 1)$.

two's complement: one's complement plus one, i.e. the sum of an n-bit number and its two's complement is 2^n (→wikipedia); numbers range from -2^{n-1} to $2^{n-1} - 1$.

excess 2^{m-1}: sum of itself and 2^{m-1}, for $m = 8$ the system is called *excess 128*

NEGATIVE BINARY NUMBERS

$(N)_{10}$	$(N)_2$	$-(N)_2$ signed mag.	$-(N)_2$ 1's compl.	$-(N)_2$ 2's compl.	$-(N)_2$ excess 128
1	0000 0001	1000 0001	1111 1110	1111 1111	0111 1111
2	0000 0010	1000 0010	1111 1101	1111 1110	0111 1110
3	0000 0011	1000 0011	1111 1100	1111 1101	0111 1101
4	0000 0100	1000 0100	1111 1011	1111 1100	0111 1100
5	0000 0101	1000 0101	1111 1010	1111 1011	0111 1011
6	0000 0110	1000 0110	1111 1001	1111 1010	0111 1010
7	0000 0111	1000 0111	1111 1000	1111 1001	0111 1001
8	0000 1000	1000 1000	1111 0111	1111 1000	0111 1000
9	0000 1001	1000 1001	1111 0110	1111 0111	0111 0111
10	0000 1010	1000 1010	1111 0101	1111 0110	0111 0110
20	0001 0100	1001 0100	1110 1011	1110 1100	0110 1100
30	0001 1110	1001 1110	1110 0001	1110 0010	0110 0010
40	0010 1000	1010 1000	1101 0111	1101 1000	0101 1000
50	0011 0010	1011 0010	1100 1101	1100 1110	0100 1110
127	0111 1111	1111 1111	1000 0000	1000 0001	0000 0001
128	nonexistent	nonexistent	nonexistent	1000 0000	0000 0000

Subtraction of binary numbers

$(N)_{10}$	$(N)_2$ 1's compl.	$(N)_2$ 2's compl.
10	0000 1010	0000 1010
+ (−3)	1111 1100	1111 1101
7	1 0000 0110	~~1~~ 0000 0111
	↑ *carry* 1	↑ *discarded*
	0000 0111	

FLOATING POINT NUMBERS

A real value x can be represented using scientific notation (base 10)

$$x = (-1)^{s} \times f \times 10^{e}$$

where s: sign, f: fraction or mantissa, e: exponent

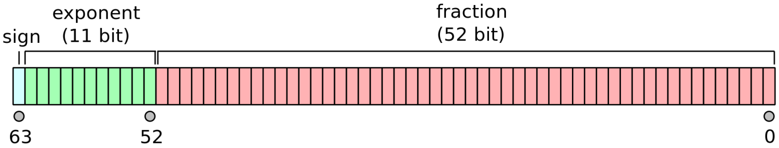
Range vs. precision:

- the **range** is determined by the number of digits in the exponent e
- the **precision** is determined by the number of digits in the fraction f

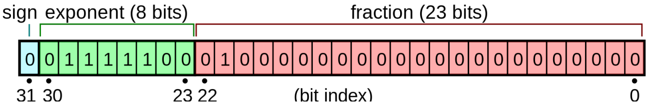
Underflow/overflow occurs, if the magnitude of x is too small/large to be represented with the given range and precision

Double precision: 64-bit representation of a real value

$$(-1)^{\text{sign}}(1.b_{51}b_{50}\ldots b_0)_2 \times 2^{e-1023} = (-1)^{\text{sign}}\left(1 + \sum_{i=1}^{52} b_{52-i}\, 2^{-i}\right) \times 2^{e-1023}$$



Single precision: 32-bit representation of a real value



from: wikipedia and wikipedia

FLOATING POINT NUMBERS

The IEEE floating-point format

Item	Single precision	Double precision
Bits in sign	1	1
Bits in exponent	8	11
Bits in fraction	23	52
Bits, total	32	64
Exponent system	Excess 127	Excess 1023
Exponent range	−126 to +127	−1022 to +1023
Smallest normalized number	2^{-126}	2^{-1022}
Largest normalized number	approx. 2^{128}	approx. 2^{1024}
Decimal range	approx. 10^{-38} to 10^{38}	approx. 10^{-308} to 10^{308}
Smallest denormalized number	approx. 10^{-45}	approx. 10^{-324}

from: A. S. Tannenbaum & T. Austin: Structured Computer Organization, 6th edition

FLOATING POINT NUMBERS

Overflow level (OFL): largest positive floating point number

$$\text{OFL} = (1 - 2^{-(p+1)}) \times 2^{(e_{max}+1)} \approx 2^{(e_{max}+1)}$$

Underflow level (UFL): smallest positive floating point number

$$\text{UFL} = 2^{e_{min}}$$

where e_{max}/e_{min} is the largest/smallest exponent, p is the number of bits in the fraction

if $|x| <$ UFL, then $x = 0.0$

if $|x| >$ OFL, then $x = \inf$