Høyskolen Kristiania

Digital technology

TK1104-1 22H

Lecturer: Toktam Ramezanifarkhani

Toktam.Ramezanifarkhani@kristiania.no

Toktamr@ifi.uio.no

The Binary Numbering Systems

Last week

- Introduction
- Computer and its main components
  - a man-made device that receives data in a form, processes these and produces new (and more useful) information built on the original data
  - What is this device?
  - What is in there?
  - Computer assembly
  - Laptop ...

This Session

Høyskolen Kristiania

- Data representation
- Number systems: decimal, binary, hexadecimal (octal)
- Simple calculations
- Precision and negative numbers
- (Coding / decoding (begins, we will have more))

- EXERCISE in this topic is important, calculate tasks to understand!

The point!

Høyskolen Kristiania

- The point is not in itself to count…
- We want to understand computers better…
- Computers are calculators
- Binary arithmetic is thus «computer psychology»

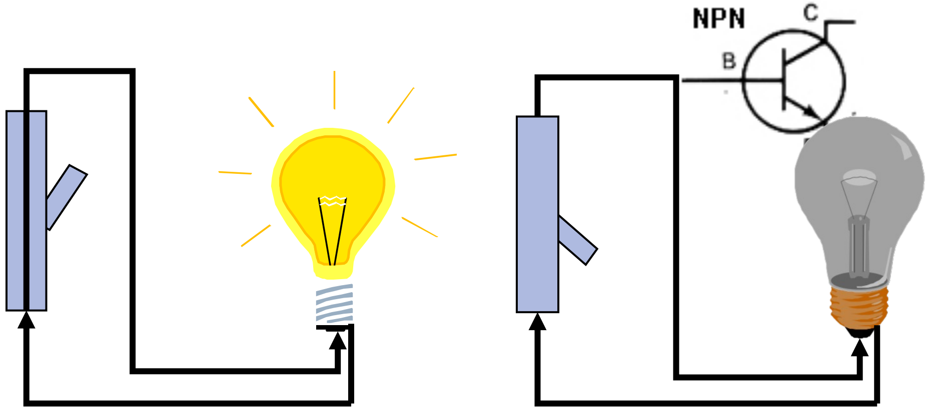
Types of data / information

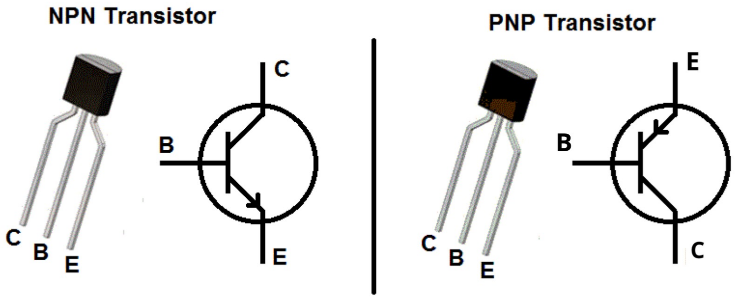
Høyskolen Kristiania

- A (general) computer processes 5 main types of information
- Numeric
  - Number calculations
- Grade-based (alphanumeric)
  - Text manipulation
- Visual
  - Pictures
- Audio
  - Sound
- Instructions
  - Internal orders to the computer (CPU) about what to do

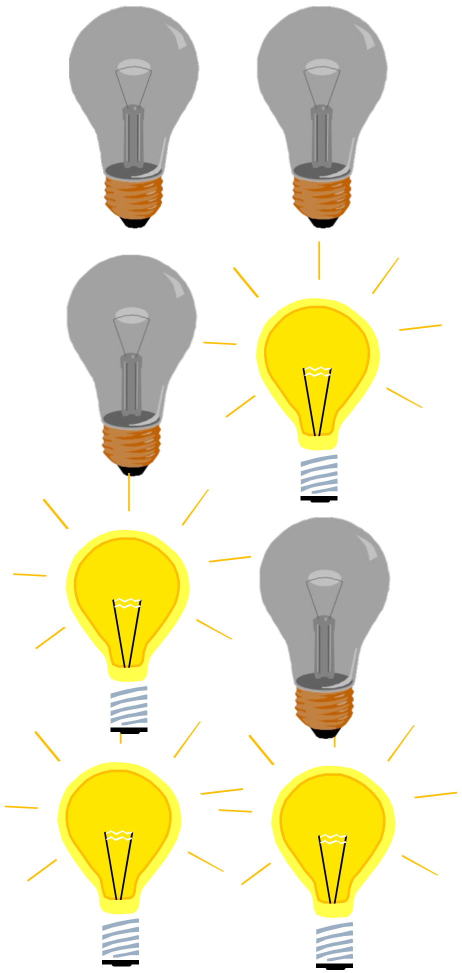
Representation in a computer

- Today's digital computers use binary numbers
- Needs only two digits: 0 and 1
- This provides the possibility of simple logic circuits
- At the same time all kinds of information can be represented in this way





Example



Im not coming

I’ll come by myself (see you!)

I’ll come a little later (wait for me)

Im coming right now

- How many messages can we send and receive by x bits(lamps)?

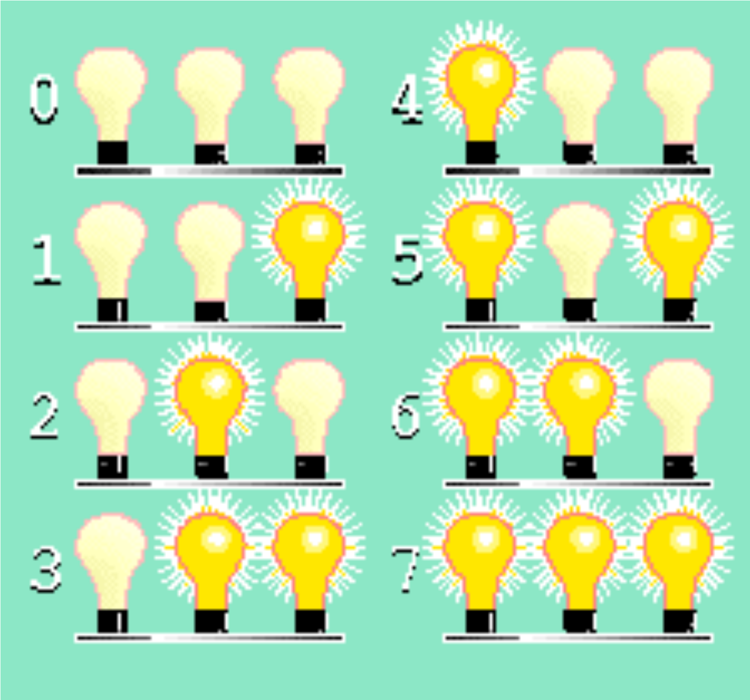
Binary number representation

- With 3 lamps we can represent $2^3 = 8$ combinations of on / off
- With Off = 0 and On = 1 we get

0 = 000	4 = 100
1 = 001	5 = 101
2 = 010	6 = 110
3 = 011	7 = 111

- This can be extended to use more lamps

(transistors), e.g. 8, 16, 32, 64, ...



Decimal numeral system or base-10

Each column indicates the number of boxes with the capacity of base^{column}

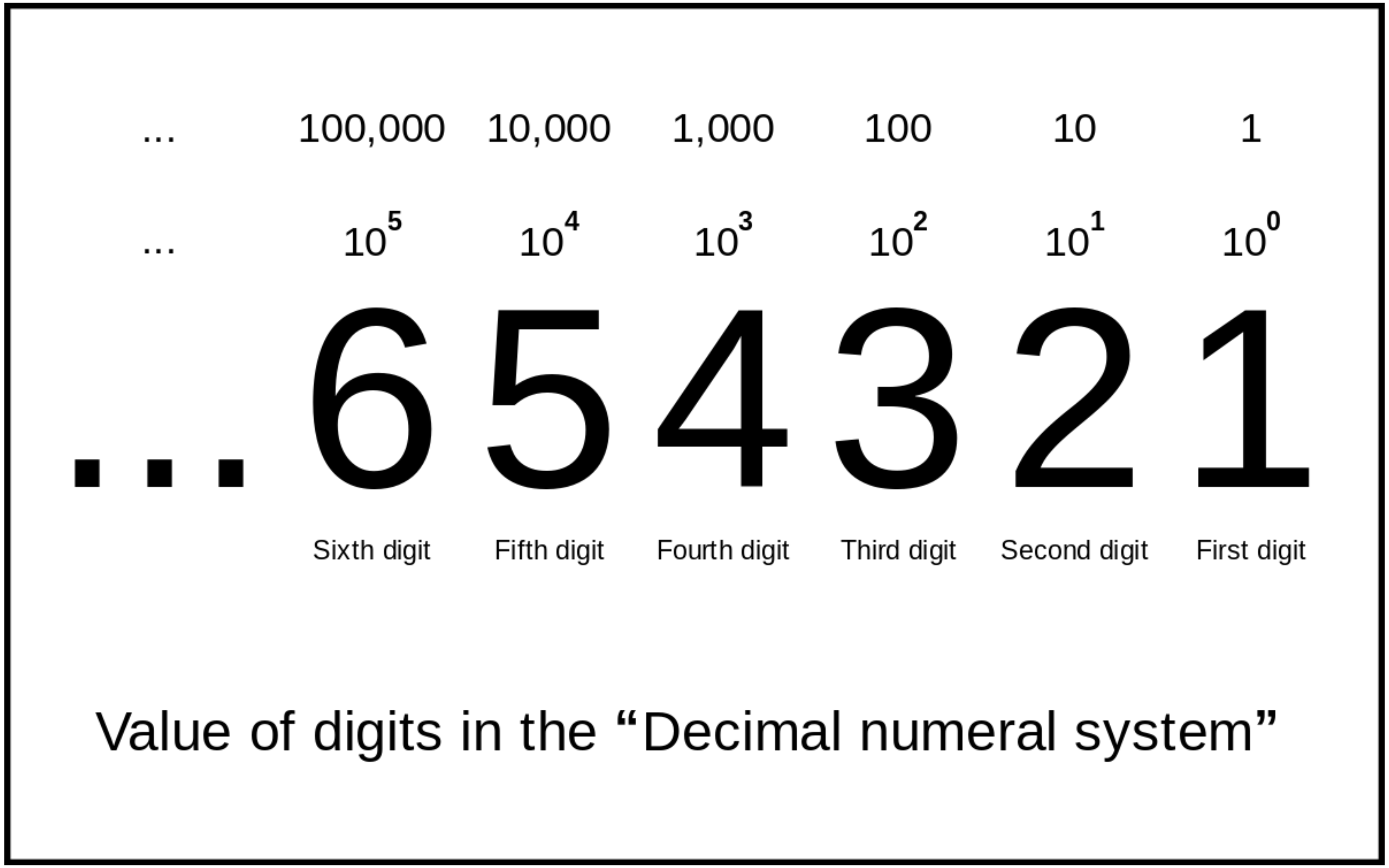
Whole Numbers

Billions			Millions			Thousands			Ones		
hundreds	tens	ones	hundreds	tens	ones	hundreds	tens	ones	hundreds	tens	ones
10^{11}	10^{10}	10^{9}	10^{8}	10^{7}	10^{6}	10^{5}	10^{4}	10^{3}	10^{2}	10^{1}	10^{0}

the decimal numeral system or base-10 positional numeral system:
a system for expressing numerals such that the position of the digit indicates the power of 10 that the digit is multiplied by to determine its value.

For example, the integer 10 has a 1 in the tens place and a 0 in the ones place.

Example



... 100,000 10,000 1,000 100 10 1
... 10^5 10^4 10^3 10^2 10^1 10^0
...654321
Sixth digit Fifth digit Fourth digit Third digit Second digit First digit
Value of digits in the “Decimal numeral system”

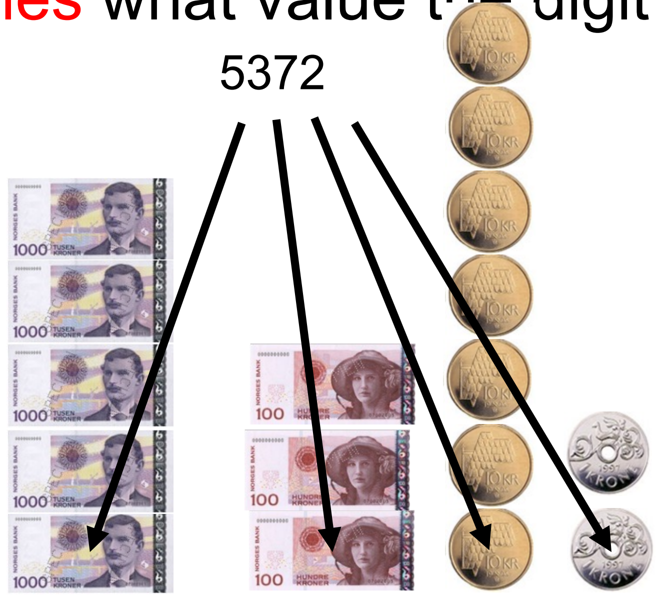
The Binary Numbering Systems

Høyskolen Kristiania

- A numbering system (base) is a way to represent numbers, base k
    - We denote the base by adding k as a subscript at the end of the number as in 1234_5 for base 5 (we can omit 10 if in base 10)
- Decimal is base 10, binary is base 2
    - We use 10 because of 10 fingers, but are interested in 2 (also 8 and 16) because computers store and process information in a digital (on/off) way
- 1 binary digit is a bit
- In 1 bit, we store a 0 or a 1
    - This doesn't give us much meaning, just 1/0, yes/no, true/false
- We group 8 bits together to store 1 byte
    - 00000000 to 11111111
    - In 1 byte, we can store a number from 0 to 255 or a character (e.g., 'a', '$', '8', ' ')

Decimal numbers - position numbers

- The "regular" (decimal) number system uses 10 digits / symbols (0 - 9), while the binary system only uses 2 digits / symbols (0-1).
- However, the principles behind the binary number system are the same as for the decimal system
- The position of a digit in a number determines what value the digit represents ("weight").



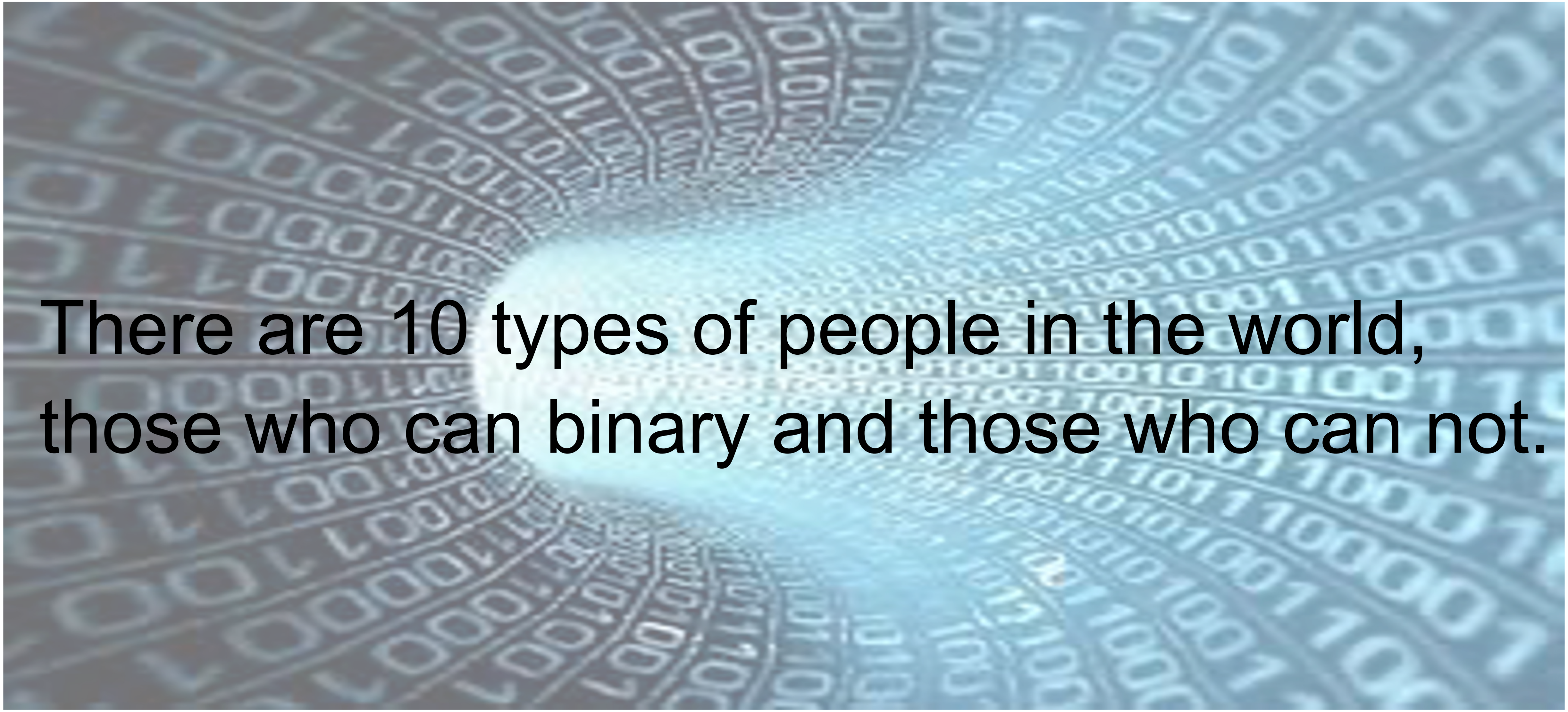
Interpreting Numbers

Høyskolen Kristiania

- The base tells us how to interpret each digit
  - The column that the digit is in represents the value base^{column}
    - the rightmost column is always column 0
  - Example:
    - 5372 in base 10 has
      - 5 in the 10^3 column (1,000)
      - 3 in the 10^2 column (100)
      - 7 in the 10^1 column (10)
      - 2 in the 10^0 column (1)
- To convert from some base k to base 10, apply this formula
  - $abcde_k = a * k^4 + b * k^3 + c * k^2 + d * k^1 + e * k^0$
  - a, b, c, d, e are the digits, k^0 is always 1
- To convert binary to decimal, the values of k^0, k^1, k^2, etc are powers of 2 (1, 2, 4, 8, 16, 32, …)

Counting with binary numbers

0	01	10	11
100	101	110	111
1000	1001	1010	1011
1100	1101	1110	1111



There are 10 types of people in the world,
those who can binary and those who can not.

Tutorials

Høyskolen Kristiania

- Binary 1 - Converting to and from Denary (decimal.)

https://www.youtube.com/watch?v=cJNm938Xwao&list=PLTd6ceoshprfijQztP-IKey4OV7nkr_va&t=129s

(https://www.youtube.com/watch?v=cJNm938Xwao&list=PLTd6ceoshprcpen2Jvs_JiuvWvqIAkzea)

Conversion from binary to decimal

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1024$

9 8 6 4 1 0

1101010011

512 256 64 16 2 1

$$1101010011 = 1*2^9+1*2^8+0*2^7+1*2^6+0*2^5+1*2^4+0*2^3+0*2^2+1*2^1+1*2^0$$

$$= 512 + 256 + 64 + 16 + 2 + 1$$

$$= 851$$

Careful! Initial zeros in 16 bit precision are missing here!!

0000 0011 0101 0011

Binary to Decimal Conversion

Høyskolen Kristiania

- Multiply each binary bit by its column value
  - In binary, our columns are (from right to left)
    - $2^0 = 1$
    - $2^1 = 2$
    - $2^2 = 4$
    - $2^3 = 8$
    - $2^4 = 16$
    - $2^5 = 32$
    - Etc
  - 10110 = $1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0 = 16 + 0 + 4 + 2 + 0 = 22$
  - 1100001 = $1 * 2^6 + 1 * 2^5 + 0 * 2^4 + 0 * 2^3 + 0 * 2^2 + 0 * 2^1 + 1 * 2^0 = 64 + 32 + 0 + 0 + 0 + 0 + 1 = 97$

Simplifying Conversion in Binary

- Our digits will either be 0 or 1
  - 0 * anything is 0
  - 1 * anything is that thing
- Just add together the powers of 2 whose corresponding digits are 1 and ignore any digits of 0
- 10110 = $2^4 + 2^2 + 2^1 = 16 + 4 + 2 = 22$
- 1100001 = $2^6 + 2^5 + 2^0 = 64 + 32 + 1 = 97$

Examples

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	1	0	1	0	1	0	1	
	=		=		=		=	
	64		16		4		1	= 85

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	1	1	1	1	0	1	1	
=	=	=	=	=		=	=	
128	64	32	16	8		2	1	= 251

11010110 =
128 + 64 + 16 + 4 + 2
= 214

10001011 = 128 + 8 + 2 + 1
= 139

11111111 = 128 + 64 + 32 +
16 + 8 + 4 + 2 + 1
= 255

00110011 = 32 + 16 +
2 + 1
= 51

Converting from Decimal to Binary

- The typical approach is to continually divide the decimal value by 2, recording the quotient and the remainder until the quotient is 0
- The binary number is the group of remainder bits written in opposite order
  - Convert 19 to binary
    - 19 / 2 = 9 remainder 1
    - 9 / 2 = 4 remainder 1
    - 4 / 2 = 2 remainder 0
    - 2 / 2 = 1 remainder 0
    - 1 / 2 = 0 remainder 1
  - 19 = 10011_2

Record the remainders and then write them in opposite order

Examples

Convert 200 to binary

200 / 2 = 100 r 0
100 / 2 = 50 r 0
50 / 2 = 25 r 0
25 / 2 = 12 r 1
12 / 2 = 6 r 0
6 / 2 = 3 r 0
3 / 2 = 1 r 1
1 / 2 = 0 r 1

200 = $(11001000)_2$

Convert 16 to binary

16 / 2 = 8 r 0
8 / 2 = 4 r 0
4 / 2 = 2 r 0
2 / 2 = 1 r 0
1 /2 = 0 r 1

16 = 10000

Convert 21 to binary

21 / 2 = 10 r 1
10 / 2 = 5 r 0
5 / 2 = 2 r 1
2 / 2 = 1 r 0
1 / 2 = 0 r 1

21 = 10101

Convert 122 to binary

122 / 2 = 61 r 0
61 / 2 = 30 r 1
30 / 2 = 15 r 0
15 / 2 = 7 r 1
7 / 2 = 3 r 1
3 / 2 = 1 r 1
1 / 2 = 0 r 1

122 = 1111010

Another Technique

Høyskolen Kristiania

- Recall to convert from binary to decimal, we add the powers of 2 for each digit that is a 1
- To convert from decimal to binary, we can subtract all of the powers of 2 that make up the number and record 1s in corresponding columns
- Example
  - 19 = 16 + 2 + 1
  - So there is a 16 (2^4), a 2 (2^1) and 1 (2^0)
  - Put 1s in the 4th, 1st, and 0th columns:
  - 19 = 10011_2

$2^0 = 1$

$2^1 = 2$

$2^2 = 4$

$2^3 = 8$

$2^4 = 16$

$2^5 = 32$

…

Conversion from decimal to binary - Example

Høyskolen Kristiania

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1024$

$$851 = 512 + 339 = 2^9 + 339$$
$$339 = 256 + 83 = 2^8 + 83$$
$$83 = 64 + 19 = 2^6 + 19$$
$$19 = 16 + 3 = 2^4 + 3$$
$$3 = 2 + 1 = 2^1 + 1$$
$$1 = 2^0$$

$$851 = 2^9 + 2^8 + 2^6 + 2^4 + 2^1 + 2^0$$

$$851 = 1*2^9 + 1*2^8 + 0*2^7 + 1*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 1*2^0$$

$851_{10} = 0000\ 0011\ 0101\ 0011_2$

More Examples

Høyskolen Kristiania

- Convert 122 to binary
  - Largest power of 2 <= 122 = 64 leaving 122 – 64 = 58
  - Largest power of 2 <= 58 = 32 leaving 58 – 32 = 26
  - Largest power of 2 <= 26 = 16 leaving 26 – 16 = 10
  - Largest power of 2 <= 10 = 8 leaving 10 – 8 = 2
  - Largest power of 2 <= 2 = 2 leaving 0
  - Done
- 122 = 64 + 32 + 16 + 8 + 2 = 1111010
- More examples:
  - 555 = 512 + 32 + 8 + 2 + 1 = 1000101011
  - 200 = 128 + 64 + 8 = 11001000
  - 199 = 128 + 64 + 4 + 2 + 1 = 11000111
  - 31 = 16 + 8 + 4 + 2 + 1 = 11111
  - 60 = 32 + 16 + 8 + 4 = 111100
  - 1000 = 512 + 256 + 128 + 64 + 32 + 8 = 1111101000
  - 20 = 16 + 4 = 10100

Number of Bits

Høyskolen Kristiania

- Notice in our previous examples that for 555 we needed 10 bits and for 25 we only needed 5 bits
- The number of bits available tells us the range of values we can store
- In 8 bits (1 byte), we can store between 0 and 255
  - 00000000 = 0
  - 11111111 = 255 (128 + 64 + 32 + 16 + 8 + 4 + 2 + 1)
- In n bits, you can store a number from 0 to 2^n-1
  - For 8 bits, 2^8 = 256, the largest value that can be stored in 8 bits is 255
  - What about 5 bits?
  - What about 3 bits?

Lets take base k and raise it to power column index e.g. K^i
Like 2 to the 3rd power = 2^3

We can have this exponential equation:
K raised to power i equals n: K^i=n   the exponent I is the logarithm of n
Then $\log^n_k$=i log base k of a number n equls p the power
i= $\log^n_x$/ $\log^k_x$.

Negative Numbers

Høyskolen Kristiania

- To store negative numbers, we need a bit to indicate the sign
  - 0 = positive, 1 = negative
- Several representations for negative numbers, we use two's complement
- Positive numbers are the same as in our previous approach
- Negative numbers need to be converted, two ways to do this:

First

- 1's complement: NOT (flip) all of the bits (1 becomes 0, 0 becomes 1)
- 2's complement: Add 1
- Example: -57 in 8 bits
  - +57 = 32 + 16 + 8 + 1 = 00111001
  - -57 = NOT(00111001) + 1 = 11000110 + 1 = 11000111

Negative number: decimal to binary

- +5=0000 0101
- -5=?
- 1's complement 1111 1010
- 2's complement= 1's complement+1
- 1111 1011=-5
- Just by changing the sign bit, the number will not be negative
-

Second

Shortcut: Starting from the right of the number

record each bit THROUGH the first 1

flip all of the remaining bits

1s become 0s, 0s become 1s

Examples (all are 8 bits)

- -57
  - +57 = 00111001
  - from the right, copy all digits through the first one:
  - -------1
  - Flip remaining bits (0011100)
  - 1100011 1 = 11000111
- -96
  - +96 = 01100000
  - from right, copy all bits through the first one:
  - --100000
  - flip rest of the bits (01)
  - 10 100000 = 10100000
- -108
  - +108 = 01101100
  - from the right, copy all digits through the first one:
  - -----100
  - flip the rest of the bits (01101)
  - 10010 100 = 10010100
- -5
  - +5 = 00000101
  - from right, copy all bits through the first one:
  - -------1
  - flip rest of the bits (0000010)
  - 1111101 1 = 11111011

Negative number: binary to decimal

There are two ways: First

Shortcut:

- $1111\ 1011 = ?_{10}$
- Find the first 10 from the left and keep the 1

  1111 1011

  -2^3
- Add the rest

  $-2^3+2^1+2^0$

  $= -5$

When you are going to convert a binary number to decimal, and it is stated that a double complement has been used on it, you will first look at the most significant bit, and decide whether it is positive or negative. If it is 0, the number is positive, and if it is 1, the number is negative. (Here you must also make sure that you know the [illegible]sion)

Example:

Say we have the number 1010 01[illegible]ecision on 2's complement. Then we have t[illegible]umber 8 places from the right, and decide if the numbe[illegible] positive. We see that the 'Sign' bit is a 1, which me[illegible]umber is negative. The first bit corresponds to th[illegible]28.

All the other bits ca[illegible] considered positive, and thus we can add them with -128 as follows:

1010 0101 = (-128) + 32 + 4 + 1 = -91

If the most significant bit had been 0, we would have got a completely different result:

0010 0101 = (0) + 32 + 4 + 1 = 37

And if we were to run the math without a 2's complement, the result is completely different:

1010 0101 = (128) + 32 + 4 + 1 = 165

In the Lab

Negative number: decimal to binary explains the shortcut- binary to decimal

Second

- Using shortcut to convert binary to decimal, we can learn how to do opposite direction
- To convert a decimal number to binary, and it is stated that a double complement has been used on it, you will first see if it is a positive or negative number
  - If the most significant bit (MSB) is 0 is the number positive,
  - if it is 1 the number is negative. (Here you must also make sure that you know the precision).
  - If the number is negative, then we know that MSB is 1, which means that we now have to find out which one of the other bits should be 1 before giving the correct answer, "What do we have to plus -128 to get the correct answer?" .
- Example: What is -58 in binary with 8-bit precision and 2's complement?

  (-128) + 58 = 70
  70 - 64 = 6
  6 – 4 = 2
  2 – 2 = 0

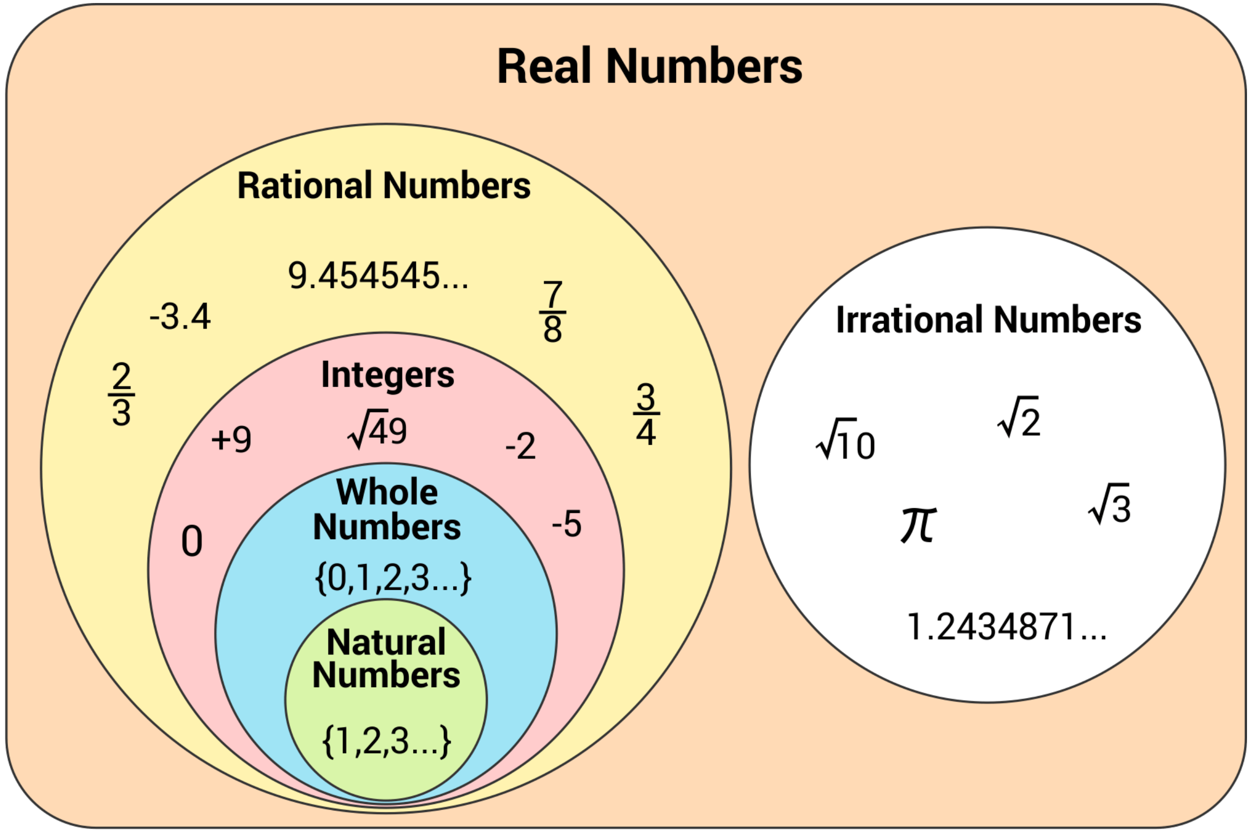
```
(-128) + 64 + 4 + 2 = -58

1 1 0 0   0 1 1 0
```

- Now we have figured out which bits should be positive (1 and not 0), these are
- 64, 4 and 2.

Decimal Fractions

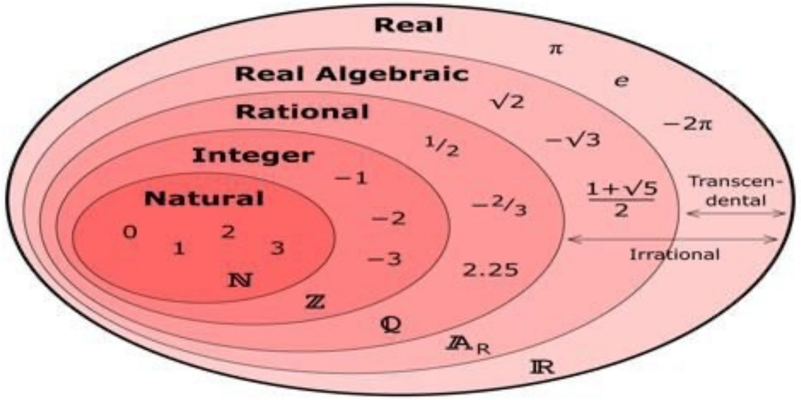
Decimal Fractions

ones	.	tenths	hundredths	thousandths	ten-thousandths	hundred-thousandths	millionths
10^0	.	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}



Real (Fractional) Numbers

- Extend our powers of two to the right of the decimal point using negative powers of 2
- $101.1 = 2^2 + 2^0 + 2^{-1}$
  - What is 2^{-1}? $1/2^1$
    - $2^{-2} = 1/2^2 = 1/4$, $2^{-3} = 1/2^3 = 1/8$
  - $10110.101 = 16 + 4 + 2 + 1/2 + 1/8 = 22\ 5/8 = 22.625$



- How do we represent the decimal point?
  - We use a *floating point* representation, like scientific notation, but in binary where we store 3 integer numbers, a sign bit (1 = negative, 0 = positive), the mantissa (the number without a decimal point) and the location of the decimal point as an exponent
    - 1011011.1 = .10110111 * 2^7
  - Mantissa = 10110111
  - Exponent = 00111 (7, the exponent)
  - Sign = 0
    - further details are covered in the text, but omitted here

Applications

Character Representations

Høyskolen Kristiania

- We need to invent a representation to store letters of the alphabet (there is no natural way)
  - Need to differentiate between upper and lower case letters – so we need at least 52 representations
  - We will want to also represent punctuation marks
  - Also digits (phone numbers use numbers but are not stored numerically)
- 3 character codes have been developed
  - EBCDIC – used only IBM mainframes
  - ASCII – the most common code, 7 bits – 128 different characters (add a 0 to the front to make it 8 bits or 1 byte per character)
  - Unicode – expands ASCII to 16 bits to represent over 65,000 characters

Hexadecimal number

Høyskolen Kristiania

- Uses 16 as a base
- Must then invent 6 "new" digits
  - $A_{16}=10$, $B_{16}=11$, $C_{16}=12$, $D_{16}=13$, $E_{16}=14$, $F_{16}=15$
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  - $11_{16} = 1*16 + 1*1 = 17_{10}$,

  $0A4C_{16} = 0*16^3 + 10*16^2 + 4*16^1 + 12*16^0$
  $0*4096 + 10*256 + 4*16 + 12*1 = 2636_{10}$

- Translates from binary to hex by grouping 4 and 4 binary numbers together ("nibbler")

  $1010\ 0011\ 1001\ 1111_2 = 0xA39F_{16}$

  $1010_2 = A_{16}$, $0011_2 = 3_{16}$, $1001_2 = 9_{16}$, $1111_2 = F_{16}$

Why Hex?

- Fewer digits–**USED TO WRITE BINARY NUMBERS MORE COMPACT**
- **Noted** in Java and many other contexts with prefix **0x**
  - Example: **0x**7F = 127 = **0b**0111 1111
- It’s so easy to calculate wrong with Hex,
- so we prefer to go binary!

ASCII (7 bit)

Høyskolen Kristiania

A tool
HextEdit

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

ASCII (notice!)

- ”A” = 0x41 -> 0100 0001
  ”a” = 0x61 -> 0110 0001
  - A little difference between uppercase and lowercase letters
- The numbers are coded with: 0x30-0x39
- Spaces are 0x20
- *More about this and other formats in the next lecture*

Example

- To store the word “Hello”
  - H = 72 = 01001000
  - e = 101 = 01100101
  - l = 108 = 01101100
  - l = 108 = 01101100
  - o = 111 = 01101111
- Hello = 01001000 01100101 01101100 01101100 01101111
- How much storage space is required for the string
  - R U 4 Luv?
- 10 bytes (5 letters, 3 spaces, 1 digit, 1 punctuation mark)
  - The ‘U’ and ‘u’ are represented using different values
    - ‘U’ = 01010101
    - ‘u’ = 01110101
  - The only difference between an upper and lower case letter is the 3rd bit from the left
    - upper case = 0, lower case = 1

Interpretation of binary codes (a few)

	0011 1110	0010 0000	0111 0010	0011 0111
(hexadesimal)	0x3E	0x20	0x72	0x37
32-bit integer	1 042 313 783			
16-bit integer	15 904		29 239	
32 bit floating point number	0.156686			
BCD	Impossible!	20	72	37
IPv4-address	62.32.114.55			
ASCII	>	space	r	7
Scankode(USB)	F5 	3 # 	F23	: . 
UTF-16	朓		爷	
JVM bytecode	istore_3	Istore_2	frem	Istore
X86 code	DS:	AND	JNO	AAA

Password
And a password Policy

What to consider for our security?

Select a good password ☺

Important: Password

Høyskolen Kristiania

- A common way to measure the quality of a password is **bit strength**
- Expresses the maximum number of attempts a random attacker needs to guess

("Brute force attack")

- Bit strength = lg2 (different characters possible in password=n) * number of characters in the password=L.
- • Eg. PIN code uses n=10 different characters (0-9) in L=4 positions=> bit strength = lg2 (10) *
- 4 = 3.32 * 4 = 13.28

$$H = \log_2 N^L = L \log_2 N = L \frac{\log N}{\log 2}$$

- NB! Practical device due to combinatorial explosion • 2 ^ bit strength = number of possible passwords that can be created
- **Recommended bit strength nowadays is about 80, ie about twelve letters and characters!**
- **In addition, you should of course avoid everything that is related to your own person, all common words (those found in dictionaries) etc.**

Important: Password (cont.)

- Number of passwords= 10^4 $=N^L$
- How many bits (H) we need to express these passwords?
- Number of passwords (created in binary system)=2^H

$$10^4= 2^H \qquad , H=?$$

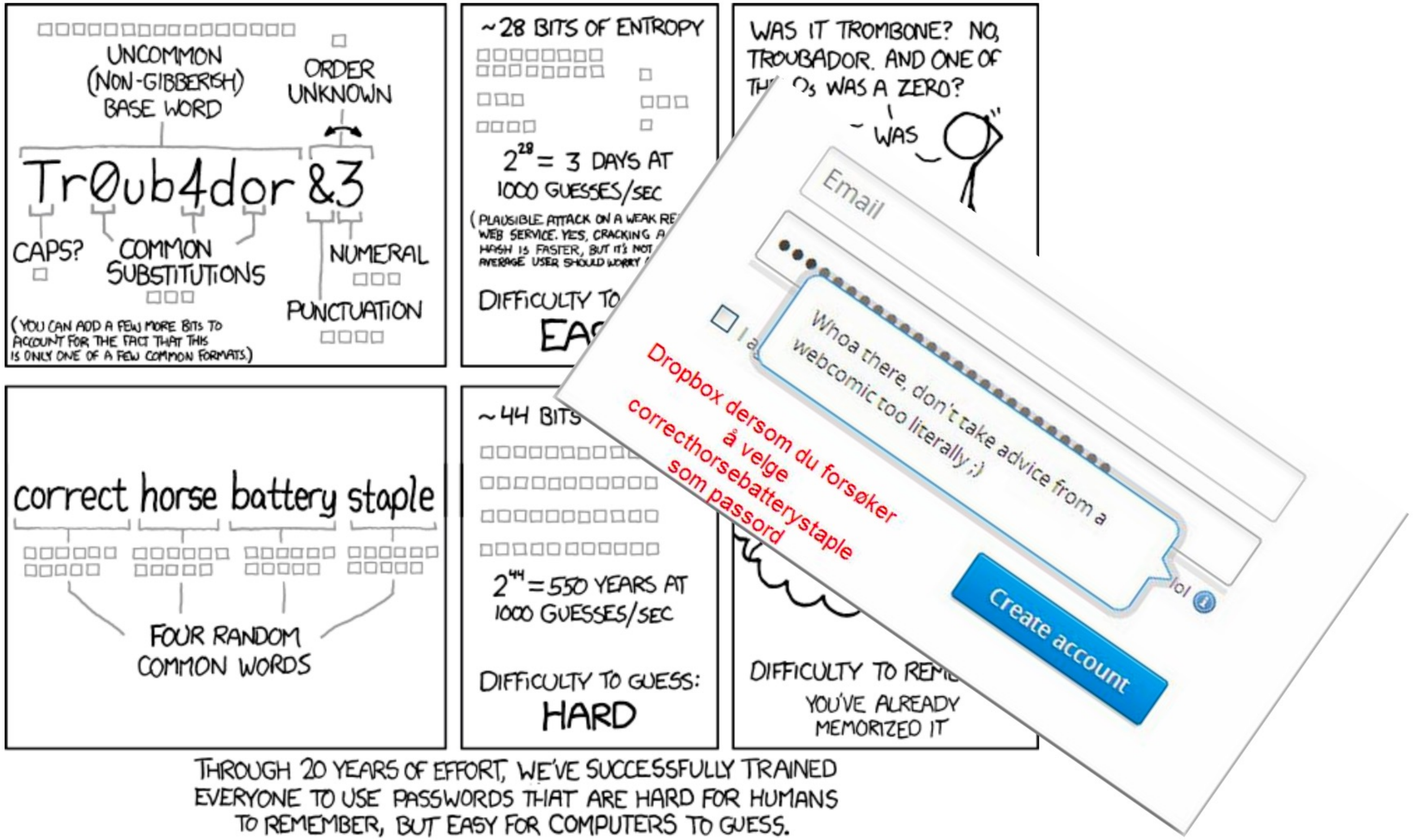
- H= number of bits=bit strength
- $H=\log_2 (10^4)=4* \log_2 10=4*3.32$

"Secure passwords"

- Should contain characters from at least three of the groups below:

Group	Example
Lowercase letters	a, b, c, …
Uppercase letters	A, B, C, …
Numerals	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Non-alphanumeric (symbols)	( ) ` ~ ! @ # $ % ^ & * - + = | \ { } [ ] : ; " ' < > , . ? /
Unicode characters	€, Γ, ƒ, and λ

XKCD: Comments



Examples of binary system

Binary Operations

- We learn the binary operations using truth tables

X	Y	AND
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	OR
0	0	0
0	1	1
1	0	1
1	1	1

X	NOT
0	1
1	0

X	Y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

- Given two bits, apply the operator
  - 1 AND 0 = 0
  - 1 OR 0 = 1
  - 1 XOR 0 = 1
- Apply the binary (Boolean) operators *bitwise* (in columns) to binary numbers as in
  - 10010011 AND 00001111 = 00000011

Examples

- AND – if both bits are 1 the result is 1, otherwise 0
  - 11111101 AND 00001111 = 00001101
  - 01010101 AND 10101010 = 00000000
  - 00001111 AND 00110011 = 00000011
- OR – if either bit is 1 the result is 1, otherwise 0
  - 10101010 OR 11100011 = 11101011
  - 01010101 OR 10101010 = 11111111
  - 00001111 OR 00110011 = 00111111
- NOT – flip (negate) each bit
  - NOT 10101011 = 01010100
  - NOT 00001111 = 11110000
- XOR – if the bits differ the result is 1, otherwise 0
  - 10111100 XOR 11110101 = 01001001
  - 11110000 XOR 00010001 = 11100001
  - 01010101 XOR 01011110 = 00001011

Binary Addition

Høyskolen Kristiania

- To add 2 bits, there are four possibilities
  - 0 + 0 = 0
  - 1 + 0 = 1
  - 0 + 1 = 1
  - 1 + 1 = 2 – we can’t write 2 in binary, but 2 is 10 in binary, so write a 0 and carry a 1
- To compute anything useful (more than 2 single bits), we need to add binary numbers
- This requires that we chain together carrys
  - The carry out of one column becomes a carry in in the column to its left

Continued

- With 3 bits (the two bits plus the carry), we have 4 possibilities:
  - 0 + 0 + 0 = 0
  - 2 zeroes and 1 one = 1
  - 2 ones and 1 zero = 2 (carry of 1, sum of 0
  - 3 ones = 3 (carry of 1 and sum of 1)
- Example:

Carry:		1	1	0	0	Initial carry in is 0
X:		0	1	1	1	
Y:	+	0	1	1	0	
Sum:		1	1	0	1	
			Carry	Carry	No Carry	

Check your work, convert to decimal!

Addition Using AND, OR, XOR

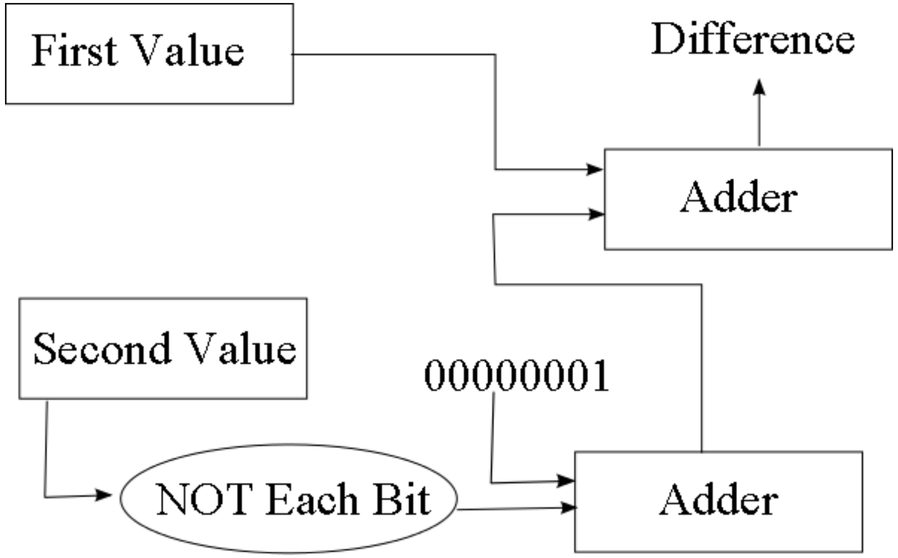
Høyskolen Kristiania

- To implement addition in the computer, convert addition to AND, OR, NOT and XOR
- Input for any single addition is two binary numbers and the carry in from the previous (to the right) column
  - For column i, we will call these X_i, Y_i and C_i
- Compute sum and carry out for column i (S_i, C_{i+1})
- S_i = (X_i XOR Y_i) XOR C_i
  - Example: if 1 + 0 and carry in of 1
    - sum = (1 XOR 0) XOR 1 = 1 XOR 1 = 0
- C_{i+1} = (X_i AND Y_i) OR (X_i AND C_i) OR (Y_i AND C_i)
  - Example: if 1 + 0 and carry in of 1
    - carry out = (1 AND 0) OR (1 AND 1) OR (0 AND 1) = 0 OR 1 OR 0 = 1

- Try it out on the previous problem

Subtraction

- From math, A – B = A + (-B)
- Store A and B in two's complement
- We can convert B into –B by
  - Flip all bits in B and adding 1
    - to flip all bits, apply NOT to each bit
    - to add 1, add the result to 00000001
- We build a subtracter unit to perform this



Subtraction

- Subtracting is thus always the same as adding the second complement

46	=	0010 1110	=	0010 1110
-37	=	-0010 0101	=	+1101 1011
9				1 0000 1001

Overflow,
We skip that !!
Due to 8 bit precision

Network Addresses

- Internet Protocol (IP) version 4 uses 32-bit addresses comprised of 4 octets
  - 1 octet = 8 bits (0..255)
  - Each octet is separated by a period
- The address 10.251.136.253
  - Stored as 00001010.11111011.10001000.11111101 in binary
  - Omit the periods when storing the address in the computer
- The network address comprises two parts
  - The network number
  - The machine number on the network
- The number of bits used for the network number differs depending upon the *class* of network
  - We might have a network address as the first 3 octets and the machine number as the last octet
  - The *netmask* is used to return either the network number or the machine number

Netmask Example

- If our network address is the first 3 octets, our network netmask is 255.255.255.0
  - 11111111.11111111.11111111.00000000
- AND this to your IP address 10.251.136.253
  - 11111111.11111111.11111111.00000000
  - AND 00001010.11111011.10001000.11111101
- Gives 00001010.11111011.10001000.00000000
  - or 10.251.136.0 which is the network number
- The machine number netmask is 0.0.0.255
  - What value would you get when ANDing 10.251.136.253 and 0.0.0.255?

Another Example

- In this case, the network address is the first 23 bits (not 24)
- The netmask for the network is 255.255.240.0

IP Address:	00001010 . 11111011 . 10001000 . 11111101 (10.251.136.253)
Netmask:	11111111 . 11111111 . 11110000 . 00000000 (255.255.240.0)
Network Address:	00001010 . 11111011 . 10000000 . 00000000 (10.251.128.0)

Different networks use different netmasks we will look at this in detail in chapter 12

Image Files

Høyskolen Kristiania

- Images stored as sequences of pixels (picture elements)
  - row by row, each pixel is denoted by a value
- A 1024x1024 pixel image will comprise 1024 individual dots in one row for 1024 rows (1M pixels)
- This file is known as a bitmap
- In a black and white bitmap, we can store whether a pixel is white or black with 1 bit
  - The 1024x1024 image takes 1Mbit (1 megabit)
- A color image is stored using red, green and blue values
  - Each can be between 0 and 255 (8 bits)
  - So each pixel takes 3 bytes
  - The 1024x1024 image takes 3MBytes
- JPG format discards some detail to reduce the image's size to about 1MB using *lossy* compression, GIF format uses a standard palette of colors to reduce size from 3 bytes/pixel to 1 (*lossless* compression)

Høyskolen Kristiania

For optional self-study

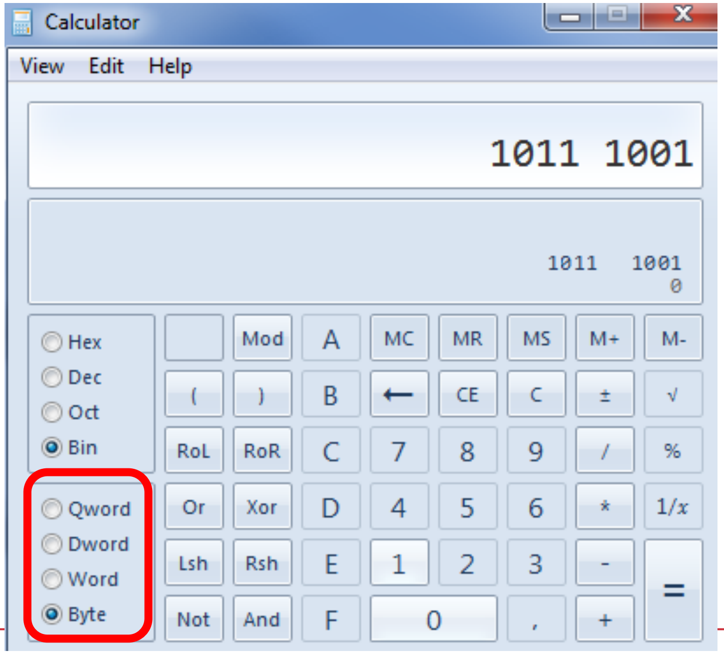
For those who want to learn some topics in more depth to better understand, here are some extra topics related to today's teaching, it must be expected some personal work to understand these topics.

There will be no questions on the exam from these, and this is therefore not considered to be part of the syllabus.

PRECISION

Høyskolen Kristiania

- In computers, everything is stored either in RAM(«memory»),in registers («memory») on the CPU or other equipment.
  - These have addresses/names
- Both in memory and on the CPU are smallest addressable device a Byte
  - It is not possible to save a bit, the minimum is 8!
- Microsoft and others operates with the devices:
  - Byte: 8 bit
  - Word: 16 bit
  - Dword: 32 bit
  - Qword: 64 bit



Calculation operations - binary numbers

Høyskolen Kristiania

Addition

```
   1 1   1
  0000 1011
+ 0001 1010
-----------
  0010 0101
```

Multiplication

```
10 * 10
-------
     00
    10
-------
   0100
```

Remember!
Doubling is the same as adding one zero furtherest to the right…

Negative numbers 2's complement

- Inversion should not result in a difference of +0 and -0
- Use 2s complement instead of 1s complement

0001 0011	
1110 1100	1's complement is the easier of the two processes as it really only involves taking a bit that is given and flipping its values.
1110 1101	2's complement = 1's complement + 1

$$0001 \xrightarrow{\text{FLIP}} 1110 \xrightarrow{+1} 1111$$

Note that both +0 and −0 return TRUE when tested for zero

2´s Complement

Høyskolen Kristiania

- 2´s Complement only works provided a certain precision, e.g. 8 bit
  - Then, 127 (0111 1111) becomes the largest number that exists,
    -128 (1000 0000) the smallest number that exists.
  - -128 «the silly number» because there is no positive version of it.
  - All other numbers can be changed by taking the second complement.

Parity

Høyskolen Kristiania

- Errors arise when data is moved from one place to another (e.g., network communication, disk to memory)
- We add a bit to a byte to encode error detection information
  - If we use even parity, then the number of 1 bits in the byte + extra bit should always be even
- Byte = 001101001 (even number of 1s)
  - Parity bit = 0 (number of 1s remain even)
- Byte = 11111011 (odd number of 1s)
  - Parity bit = 1 (number of 1s becomes 8, even)
- If Byte + parity bit has odd number of 1s, then error
- The single parity bit can detect an error but not correct it
  - If an error is detected, resend the byte + bit
- Two errors are unlikely in 1 byte but if 2 arise, the parity bit will not detect it, so we might use more parity bits to detect multiple errors or correct an error

Big vs little Endian

Høyskolen Kristiania

- In what order should bits and bytes be stored and transmitted?
- For representations that require more than one byte, we have two options
  - Start with least significant byte (LSB at the lowest address)
  - Start with most significant byte (MSB at the lowest address).
- In practice, this means that in UTF-16, for example, "A" has two different representations
  - Big Endian: 0x00 41
    (IMB, Mac inntil Intel)
  - Little Endian: 0x41 00
    (This was / is most common on Intel / AMD)
  - Misunderstanding will replace A with 䄀

RAM-adresse	Big Endian	Little Endian
001A3BF7		
001A3BF8	00	41
001A3BF9	41	00
001A3BFA		

A float

- Know how floating point numbers in a position number system work
- Know the coding standard IEEE 754
- Know the rounding issues associated with the use of floating point numbers

68

A floating-point number

Høyskolen Kristiania

- How do you represent 5,625 binary?
- As in any other position number system?

  $5.625 = 5\ 5/8 = 4 + 1 + 1/2 + 1/8 =$

  $1*2^2 + 0*2^1 + 1*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3}$

  $= 101{,}101$

- This is exactly the equivalent of 103.57 being a notation for:
  $1*10^2 + 0*10^1 + 3*10^0 + 5*10^{-1} + 7*10^{-2}$
- ie Converting to "comma numbers" is exactly the same as what we do in the decimal number system

Ex: Convert 0.6875 to bin

Høyskolen Kristiania

Start numbers	Double number	Result so far
0.6875	1.375	.1
0.375	0.75	.10
0.75	1.5	.101
0.5	1.0	**.1011**

IEEE 754 og x87

- The IEEE 754 standard was developed by W. Kahn in conjunction with Intel's development of the 8087 math coprocessor
- Used in most computers. Intel math processors (built into all CPUs after Pentium).
- IEEE defines "single precision" (used by float in Java, 32 bit) and "double precision" (double in Java, 64 bit).
- Intel's math "coprocessor" also has a third, higher precision: "extended precision" (80 bit, always used for intermediate calculation in the floating point unit (FPU)).
- The latest version of the standard is IEEE 754-2008
- Also allows 128 bit floating point numbers (34 decimal places) m.m.

IEEE single precision

31	30 … 23	22 … 0
s	exp	mantisse

- Exactly up to about 7 decimal places.
- s is a character bit. 0 for positive, 1 for negative.
- exp (8 bit) er “biased” = real exponent + 7Fh. The values 00h and FFh have special meanings.
- Mantissa is 23 bits - the first 23 bits after 1 in the significant end.

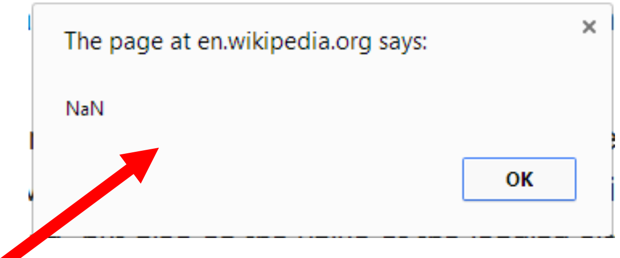
Ex: 5,8 i IEEE single precision?

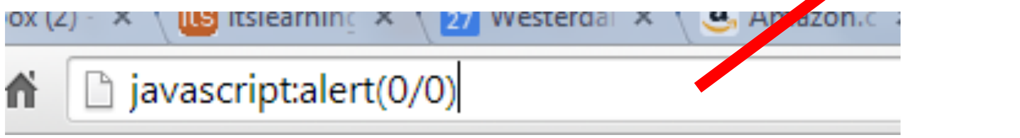
- 5,8 = 101,1100 1100 1100 1100 1100 1100 …
- With 23 numbers after the decimal point, we have:
  1.0111 0011 0011 0011 0011 001 $*2^{10}$
- The sign bit is positive = 0.
- Adjusted exponent is 0x7F + 0x02 = 0x81
- We then get:
  0100 0000 1011 1001 1001 1001 1001 1001
  (sign and mantissa are underlined)
  or 40B99999h
- Note: In C, 5.8 is represented as 40B9999A, because we dropped an LSB that was a 1. There is a better approach to 5.8

Ex: What is -0.5?

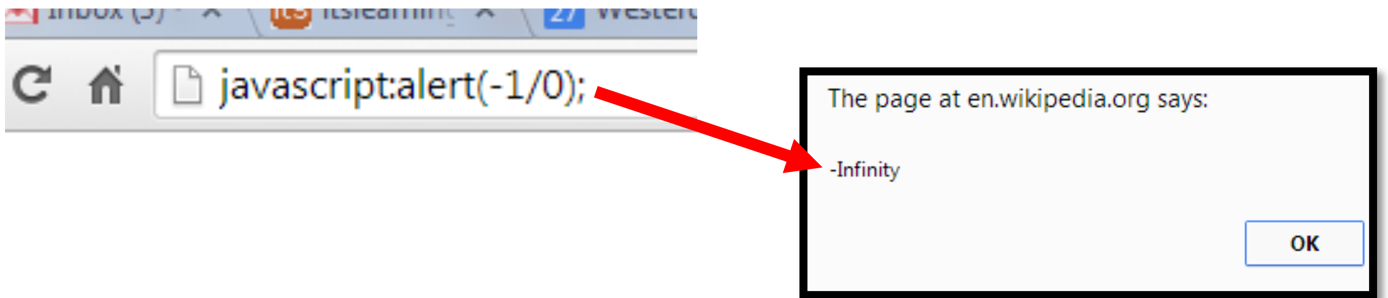
- $0.5_{10} = 0.1_2$
- With 23 digits after the decimal point, we have $1.00000000000000000000000 * 2^{-1}$
- The sign bit is negative = 1.
- The exponent is 7Fh + -1h = 7Eh
- 1011 1111 0000 0000 0000 0000 0000 0000 (BF000000h)

Not a Number





- IEEE754 has its own codes for results that are not ordinary numbers (f111 1111 1kxxx …): NaN
- This is available in several formats:
  - Depending on whether the code is to be used further or not(k=1 er «quiet NaN»)
  - Depending on the type of (mis) calculation that caused NaN
  - There are also separate codes for positive and negative infinity etc..



A float point numbers can be dangerous!!

- An Ariadne rocket crashed due to a floating point error
- For example. 10% = 0.1 in the decimal number system
  - Binary it is not possible to write 1/10 with a final number of digits, it will be 0,000110011001100110011001100…
  - If we convert back to decimal numbers, we can end up with 1/10=0,099999994
  - In IEEE754 you can specify the type of rounding to be done in which direction, but still always have to consider how many digits you can actually trust.

Code tables

Høyskolen Kristiania

- **BCD** – Binary Coded Decimals
  - Each digit in the decimal number gets its 4-bit binary code
  - 529_{10} = 010100101001_{BCD}
  - Also has other variants where we use 8 bits per decimal digit, e.g.
  - 529_{10} = $0000101\ 00000010\ 00001001_{PBCD}$
  - Intel / AMD processors still have their own instructions for BCD calculations
- Similar to alphanumeric coding
  - All digits, letters (lower and upper case) and special characters are numbered
  - EBCDIC, ASCII, ISO, Unicode
  - $M = 77_{ASCII}$, $; = 94_{EBCDIC}(5E_{16})$