Lecture 6

Data representation

Number Systems

Integers, Floating Points

Characters, Strings

Sounds, Images, Other data

Multi-byte data

Programs

Structured data

Types of Data

- Types of data for communicating with humans
  - Images, videos, sounds, characters, …
- Types of data stored in system
  - Integers, floats, characters, strings, booleans
  - Programs
  - Image and video formats, sound formats, packing standards, ...
- Types of data understood by processor
  - There are machine instructions for this type of data
  - Integers
  - Floats (most processors now)
  - Booleans (some processors, not usually)
  - Characters or strings (some processors, not usually)
  - Machine instructions

Data representation

- Question: how to represent various types of data?
- Answer: code them into bits
  - All data in system is in bits
- All processed data has its own coding methods
  - All coding methods are not standardized
  - There may be many coding systems for any type of data
    - Integers, floats, characters, strings, images, videos, sounds, …
  - Problem: do systems/machines understand each other?
    - Data representation may need to be changed, when data is copied from one system to another

Representing Data in System

- All data in in binary bits (0 or 1)
  - Binary digits: 0, 1
  - Easy to implement in electronic circuits
  - Easy to design and optimize logic with Boole's algebra
- Memory composed of equal size words — sana
  - word = 32 bits (earlier 16, 32, 48, or 64 bits, or …)
- Word is composed of equal size (8 bit) bytes — tavu

word

	byte	byte	byte	byte
bits:	1000 1101	1010 1101	1011 1100	1111 0001
hex:	8 D	A D	B C	F 1

Data Representation for CPU

- All data is coded into bits
- In memory all data can be represented in any coding system (representation) agreed upon
- Some representations are understood by the processor (I.e., processor understands them)
  - Integers and floating points (almost always)
  - Truth values, characters, strings (sometimes)
  - Images, videos, sounds (usually not, unless specialized processor)

TTK-91: integers

- Processing other data types is done with software (i.e., many instructions, subroutine, or method)
  - E.g., characters (their encoding) can be processed with integer instructions or subroutines using them
  - Rational numbers, 128-bit integers (?), large arrays, records, objects, fingerprints, sounds, images, videos, smells, …

Number systems

- From binary to decimal

  10110011 → 179

- From decimal to binary

  179 → 10110011

- From binary to hexadecimal

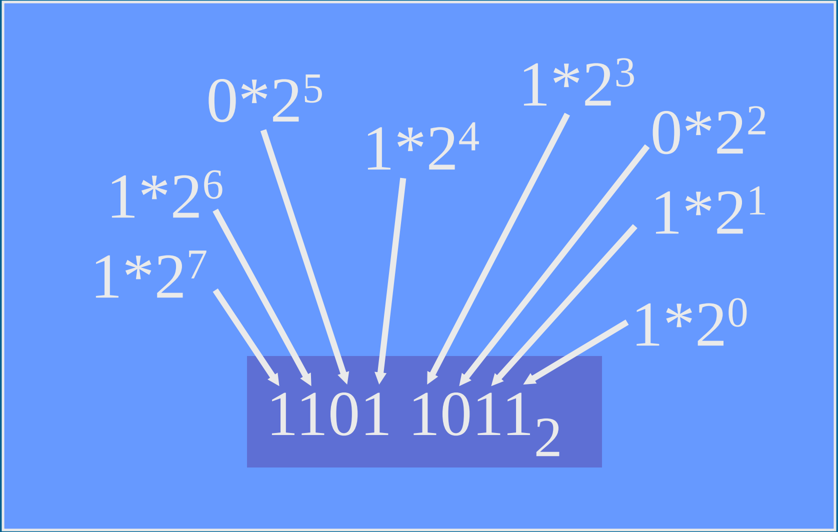
  10110011 → B3 (tai 0xB3)

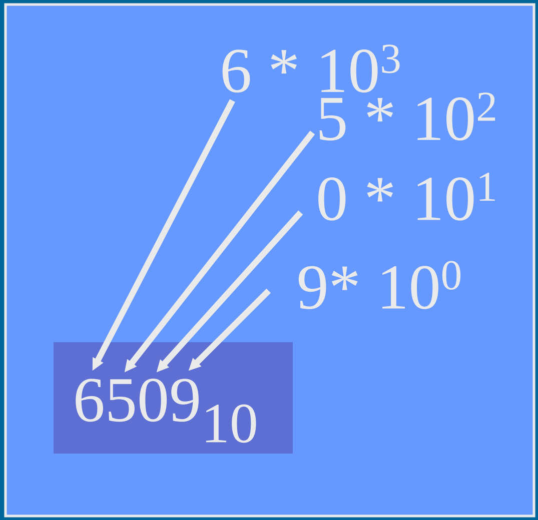
- From hexadecimal to binary

  0xB3 = B3 → 10110011

Binary system

- Base 2, digits 0 and 1
  - Digit weights from right to left: $1=2^0$, $2=2^1$, $4=2^2$, $8=2^3$, $16=2^4$, $32=2^5$, …
  - In decimal system the weights are $1=10^0$, $10=10^1$, $100=10^2$, $1000=10^3$, …





Binary examples (9)

+32 +16 +8 +1

0011 1001 = ? = 57_{10}

+2 +1

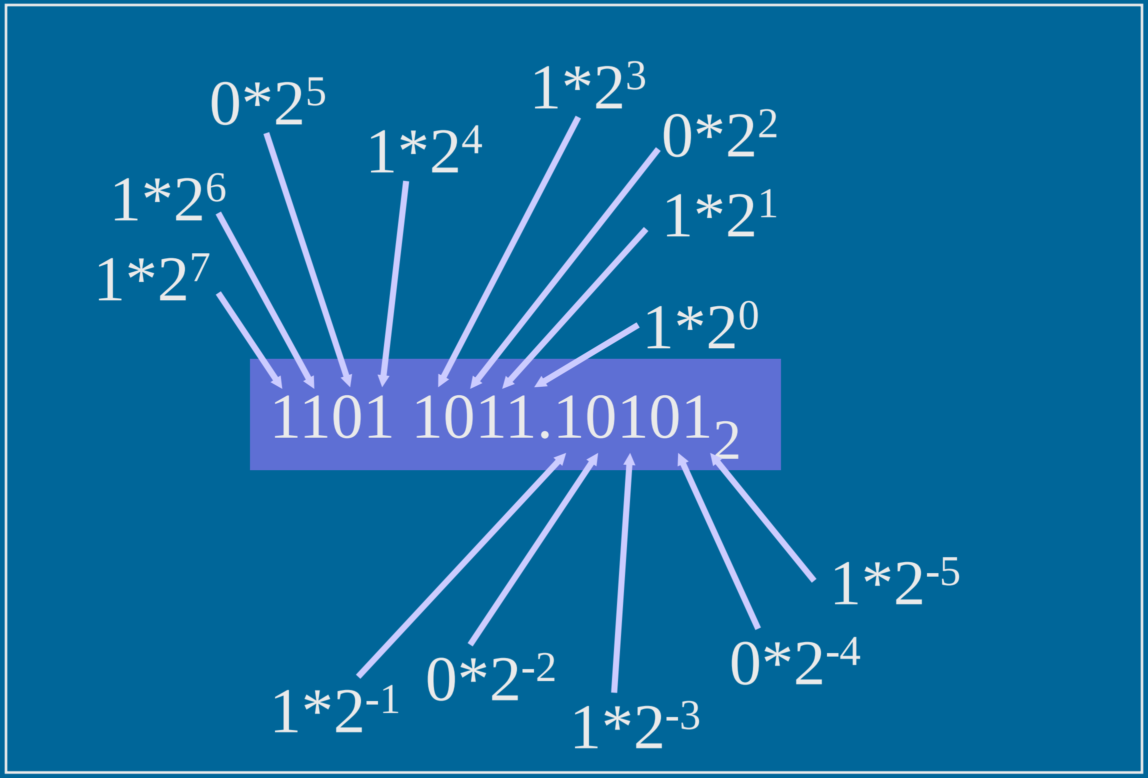
0000 0011 = ? = 3_{10}

+64 +16 +4 +1

0101 0101 = ? = 85_{10}

Binary Part and Binary Point

- Binary numbers may also have a binary part (fractional part), just like decimal numbers may have a decimal part



Binany part examples

+4 +1 +0.5 = 2^{-1}

+0.125 = 2^{-3}

0101.101 = ?

= 5.625_{10}

+4 +2 +0.125 = 2^{-3}

+0.0625 = 2^{-4}

0110. 0011 = ?

= 6.1875_{10}

Changes in Number System Representations

- Base 2 system $\Rightarrow$ base 10 system
  - Given earlier
- Base 10 system $\Rightarrow$ base 2 system
  - Do integer and decimal parts separately
  - Integer part:
    - Divide continuously by 2, until remainder is 0 jäljellä
    - Take remainders in reverse order

Decimal ⇒ Binary Integer Example (11)

$57_{10} = ?$

57/2 = 28 rem 1

28/2 = 14 rem 0

14/2 = 7 rem 0

7/2 = 3 rem 1

3/2 = 1 rem 1

1/2 = 0 rem 1

done

$= 11\ 1001_2$

$= 0011\ 1001_2$

Decimal ⇒ Binary
Desimal part ⇒ Binary part

- Multiple decimal part repeatedly by 2, until
  - Desimal part = 0 (exact binary part)
  - Enough bits for sufficient accuracy
- Result is given by taking the integer parts (0 or 1) from multiplied decimal parts in computed order

Decimal ⇒ Binary

Desimal Part ⇒ Binary Part example

$0.1875_{10} = ?$

2 * 0.1875 = 0.375	=	0	+ 0.375
2 * 0.375 = 0.75	=	0	+ 0.75
2 * 0.75 = 1.5	=	1	+ 0.5
2 * 0.5 = 1.0	=	1	+ 0.0

done

$= 0.0011_2$

$= 0.00110000000000000000_2$

Decimal ⇒ Binary

Desimal Part ⇒ Binary Part example 2

$0.3_{10} = ?$

2 * 0.3	= 0.6	= 0	+ 0.6
2 * 0.6	= 1.2	= 1	+ 0.2
2 * 0.2	= 0.4	= 0	+ 0.4
2 * 0.4	= 0.8	= 0	+ 0.8
2 * 0.8	= 1.6	= 1	+ 0.6
2 * 0.6	= 1.2	= 1	+ 0.2
2 * 0.2	= 0.4	= 0	+ 0.4
2 * 0.4	= 0.8	= 0	+ 0.8
2 * 0.8	= 1.6	= 1	+ 0.6
	done	↓	

$= 0.010011001\ldots_2$

$= 0.0100110011001100110011001_2$

$= 0.0\underline{1001}_2$

Hexadecimal Representation

- Binary numbers are necessary, but they are difficult to read/write for humans
  - Too many digits
- Write them down as hexadecimal numbers
- 4 bits is always one hexadecimal digit
- One base 16 number is always 4 bits
- Base 16 digits are:
  0,1,2,3,4,5,6,7,8,9, A, B, C, D, E ja F

A	B	C	D	E	F
10	11	12	13	14	15

Hexadecimal Examples

binary: 0100 0111 1001 1010 1111

hexa: 4 7 9 A F

$= 479AF_{16}$

$= 0004\ 79AF_{16}$ = 0x 479AF

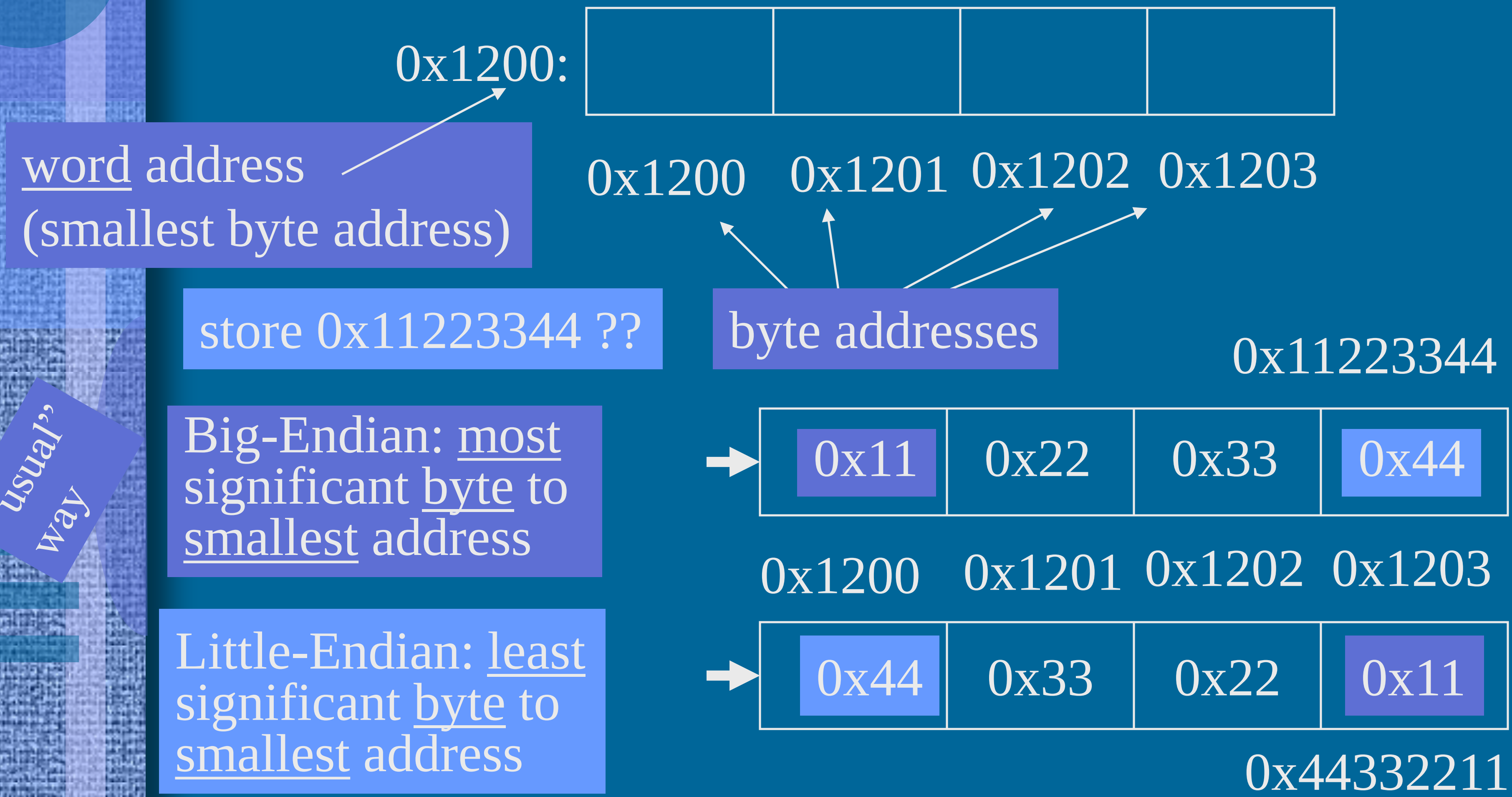
hexa: $120ADF_{16}$

1 2 0 A D F

binary: 0001 0010 0000 1010 1110 1111

Big Endian vs. Little Endian

- How to store multi<u>byte</u> data?
  - Problem concerns usually only (double) words



Big-Endian: most significant byte to smallest address

Little-Endian: least significant byte to smallest address

Integers

value representation

+57 = 0011 1001

Positive numbers usually directly binary

- Signed integers (sign magnitude)

sign bit = MSB = most significant bit

value -57 = 1011 1001 repres.

- One's complement

-57 = 1100 0110

"sign" bit

- Two's complememt

+1

-57 = 1100 0111

"sign" bit

- Biased representation
  - E.g., add 127 (=2^7 –1)
    - Usually add $2^{nrBits-1}$ -1
  - Store unsigned

-57 = 0100 0110

-57 + 127 =70

+57 = 1011 1000

+57+127=184

Floating Point Values

- Correspond to real numbers in math
- Compromise
  - Range vs. accuracy
- Always fixed accuracy
  - Same number of significant digits
- Sign, significant digits, magnitude

+5678901.2345678 vs. $+5.678901 * 10^{6}$

-0.000012345678 vs. $-1.23456\underline{8} * 10^{-5}$

+111.1010101111 vs. $+1.1110101\underline{1} * 2^{2}$

IEEE 32-bit FP Standard

$+6144.0_{10} = +1\ 1000\ 0000\ 0000.0_2 = +0.0011 * 2^{15} = \ldots$

biased +127

(+0x7F)

1b	8b (+127)	23b
"+"	"15"	"0.1875" = "0.0011"
sign	exponent	**mantissa** or significand

$1/8 = 0.1250$
$1/16 = 0.0625$
0.1875

0 = '+'
1 = '-'

- 23 bits for mantissa so that ...

1) Binary point (.) is immediately after 1st bit (bit 1)

2) Mantissa is <u>normalized</u>: leftmost bit is one (1)

3)Leftmost (most significant) bit (=1) <u>is not stored</u> (it is implied "hidden" bit)

mantissa	eksponent
0.0011	"15"
1.1000	"12"
1000	"12"

24 bit mantissa in 23 bit data field!

Miksi käytetään piilobittiä?

IEEE 32-bit FP Values

$23.0 = +10111.0 * 2^0 = +1.0111 * 2^4 = ?$

4+127=131 | 0x41B4 0000

sign	exponent	mantissa or significand
0	1000 0011	011 1000 0000 0000 0000 0000
1 bit	8 bits	23 bits

$1.0 = +1.0000 * 2^0 = ?$

0+127 = 127 | 0x3F80 0000

sign	exponent	mantissa or significand
0	0111 1111	000 0000 0000 0000 0000 0000
1 bit	8 bits	23 bits

IEEE 32-bit FP Values

0x40780000

0	1000 0000	111 1000 0000 0000 0000 0000
sign 1 bit	exponent 8 bits	mantissa or significand 23 bits

X = ?

$X = (-1)^0 * \mathbf{1}.1111 * 2^{(128-127)}$

$= 1.1111_2 * 2$

$= (1 + 1/2 + 1/4 + 1/8 + 1/16) * 2$

$= (1 + 0.5 + 0.25 + 0.125 + 0.0625) * 2$

$= 1.9375 * 2$

$= 3.875$

$= 11.111_2$

$= 2 + 1 + 1/2 + 1/4 + 1/8$

$= 3.875$

Discuss

IEEE Standard, Special Cases

- +/- 0 — 0/1 0000 0000 0…0
- +/- ∞ — 0/1 1111 1111 0…0
- Quiet NaN — 0/1 1111 1111 1?..?1?..?
- Signaling NaN — 0/1 1111 1111 0?..?1?..?

?..?1?..?
(= not all zeroes)

- Very small, not normalized (subnormal) numbers
  - Exponent: 2^{-126}
  - Hidden bit: 0

0/1 0000 0000 ??..?1?..?

UCS and Unicode

- UCS - Universal Character Set
- Same chatacter sets, different standards
- 2 bytes (16 bits) per character
  – 65536 characters for some 200000 symbols used in the world
- (32-bit UCS-4 includes also all Chinese characters)
- Control characters
  – 0x0000-001F and 0x0080-009F
  – 0x007F = DELETE, 0x0020 = SPACE
- UCS has also shorter 8-bit ”lines” of code
  – Different regions may have their own 8-bit codes, e.g., UTF-8

(Character) Strings

- Usually sequential set of bytes
- Need to code length somehow:
  – Special character at end (extra byte!)
    - C language: ’\0’ = 0x00
  – Use record

20	”Usually not any more!”
length	string

- Usually not own machine instructions (any more)
- Manipulate with subroutines
    - Integer and bit manulation instructions
    - Some (older) machine have ”strcopy” and ”strcmp” instructions
(assuming something about character set: length, bytes/char)

(Boolean) Truth Values

- Boolean TRUE and FALSE
- Usually encoded as TRUE=1, FALSE=0
  - But not always!
  - Boolean ***A and B*** = Integer ***A*B***
- Often one Boolean value per word
  - Remaining 31 bits are zeroes
  - Boolean variables in high level languages
- Sometimes packed 32 values per word
- Not own machine instructions, manipulate with bit manipulation instructions and subroutines
  - Bit manipulation instructions are usually for all bits in a word (byte)

Images

- Many image standards
  - GIF, JPEG, TIFF, BMP, ….
  - Generality, transportability, packing density
  - How much computation needed to unpack and display?
  - File header tells, which format is used
- Often packed to optimize space
  - Optimized on space (transmission time), and not on unpacking time?
  - Unpacking may take lots of processing time
- Not usually own machine instructions, manipulate with subroutines and/or display processors

Video image

- Many standards
  - MPEG (Moving Pictures Expert Group)
  - AVI (Audio Visual Interleave)
  - MOV, INDEO, FLI, GL, DVD, ...
- Not usually own machine instructions, manipulate with subroutines and/or display processors

Sounds

- Two basic approaches
  - Perfect sound data
    - 44100 samples/sec, 16 b/sample, 88KB /sec
  - Synthethized sounds
    - MIDI-instructions
      - Music Instrument Digital Interface
      - ”Play note N with loudness V”
- Not usually own machine instructions, manipulate with subroutines and/or sound/multimedia processors
  - Sound processor may be integrated with mother board or display processor

Taste, smell, feel, and other dataa

- Star brightness, boat type, attractiveness, …
- Application dependent implementation, no standards agreed upon
  - Integers (discrete data)
    - Boat type? [1, 50]?
  - Floating point values (continuous data)
    - Attractiveness? [-∞, +∞]?
- No own machine instructions, manipulate with subroutines

Machine Instructions

- Each processor type has its own
- Instruction are 1 byte or longer
  - SPARC, all instructions: 1 word, 4 bytes
  - ARM, all instructions: 1 word, 4 bytes
  - Pentium II: 1-16 bytes, many variations
- Instructions have 1 or many forms, each with varying number of fields
  - opcode, Ri, Rj, Rk, memory access mode
  - Long or short co nstant (integer)

TTK-91, all instructions: 1 word, 1 form

-- End --

Intel 4004, 1971

- Faggin, Hoff, Mazor
- 1st microprosessor
- Size 3x4 mm, $200
- 2300 transistors
- 4 bit word
- Designed for calculator
- Same computational power as Eniac (18000 vacuum tubes)

Busicom 141-PF