



NATIONAL RESEARCH
UNIVERSITY



Computer Architecture and Operating Systems

Lecture 2: Data Representation

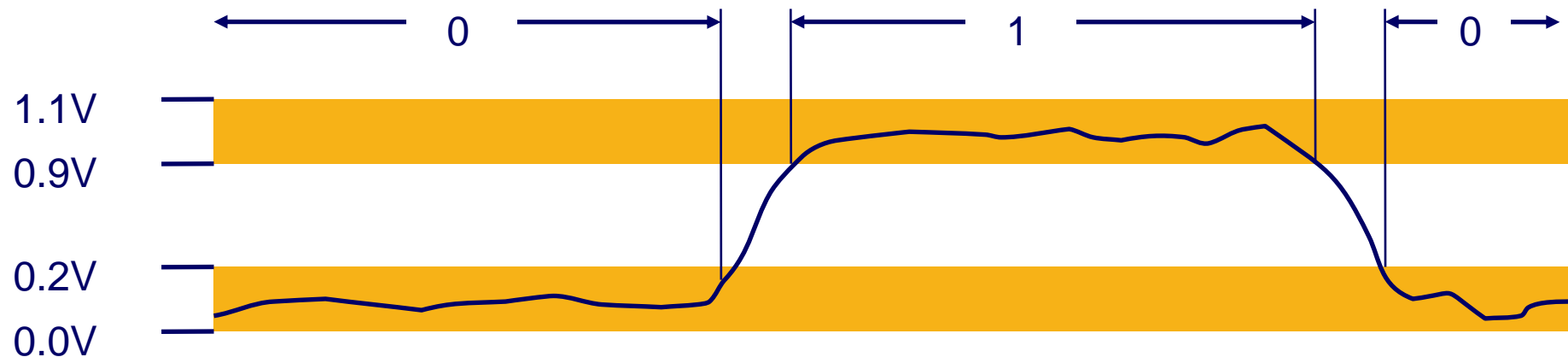
Andrei Tatarnikov

atatarnikov@hse.ru

[@andrewt0301](#)

Everything is Bits

- Each bit is 0 or 1
- By encoding/interpreting sets of bits in various ways
 - Computers determine what to do (instructions)
 - ... and represent and manipulate numbers, sets, strings, etc...
- Why bits? Electronic implementation
 - Easy to store with bistable elements
 - Reliably transmitted on noisy and inaccurate wires



Number Systems

■ Decimal numbers

1's column
10's column
100's column
1000's column

$$5374_{10} = 5 \times 10^3 + 3 \times 10^2 + 7 \times 10^1 + 4 \times 10^0$$

five thousands three hundreds seven tens four ones

■ Binary numbers

1's column
2's column
4's column
8's column

$$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13_{10}$$

one eight one four no two one one

Powers of Two

- $2^0 = 1$
- $2^1 = 2$
- $2^2 = 4$
- $2^3 = 8$
- $2^4 = 16$
- $2^5 = 32$
- $2^6 = 64$
- $2^7 = 128$
- Handy to memorize up to 2^{10}
- $2^8 = 256$
- $2^9 = 512$
- $2^{10} = 1024$
- $2^{11} = 2048$
- $2^{12} = 4096$
- $2^{13} = 8192$
- $2^{14} = 16384$
- $2^{15} = 32768$

Number Conversion

- Decimal to binary conversion:
 - Convert 10011_2 to decimal
 - $16 \times 1 + 8 \times 0 + 4 \times 0 + 2 \times 1 + 1 \times 1 = 19_{10}$
- Decimal to binary conversion:
 - Convert 47_{10} to binary
 - $32 \times 1 + 16 \times 0 + 8 \times 1 + 4 \times 1 + 2 \times 1 + 1 \times 1 = 101111_2$

Binary Values and Range

- N -digit decimal number
 - How many values? 10^N
 - Range? $[0, 10^N - 1]$
 - Example: 3-digit decimal number:
 - $10^3 = 1000$ possible values
 - Range: $[0, 999]$
- N -bit binary number
 - How many values? 2^N
 - Range: $[0, 2^N - 1]$
 - Example: 3-digit binary number:
 - $2^3 = 8$ possible values
 - Range: $[0, 7] = [000_2 \text{ to } 111_2]$

Hexadecimal Numbers

- Base 16
- Shorthand for binary

Hex Digit	Decimal Equivalent	Binary Equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Hexadecimal to Binary Conversion

- Hexadecimal to binary conversion:
 - Convert $4AF_{16}$ (also written 0x4AF) to binary
 - $0100\ 1010\ 1111_2$
- Hexadecimal to decimal conversion:
 - Convert $4AF_{16}$ to decimal
 - $16^2 \times 4 + 16^1 \times 10 + 16^0 \times 15 = 1199_{10}$

Bits, Bytes, Nibbles...

- Bits

10010110

most significant bit least significant bit

- Bytes & Nibbles

byte

10010110

nibble

- Bytes

CEBF9AD7

most significant byte least significant byte

Encoding Byte Values

- Byte = 8 bits
 - Binary 00000000_2 to 11111111_2
 - Decimal: 0_{10} to 255_{10}
 - Hexadecimal 00_{16} to FF_{16}
 - Base 16 number representation
 - Use characters '0' to '9' and 'A' to 'F'
 - Write $FA1D37B_{16}$ in C as
 - `0xFA1D37B`
 - `0xfa1d37b`

Example Data Representations

C Data Type	Typical 32-bit	Typical 64-bit
char	1	1
short	2	2
int	4	4
long	4	8
float	4	4
double	8	8
long double	–	–
pointer	4	8

Byte-Oriented Memory Organization

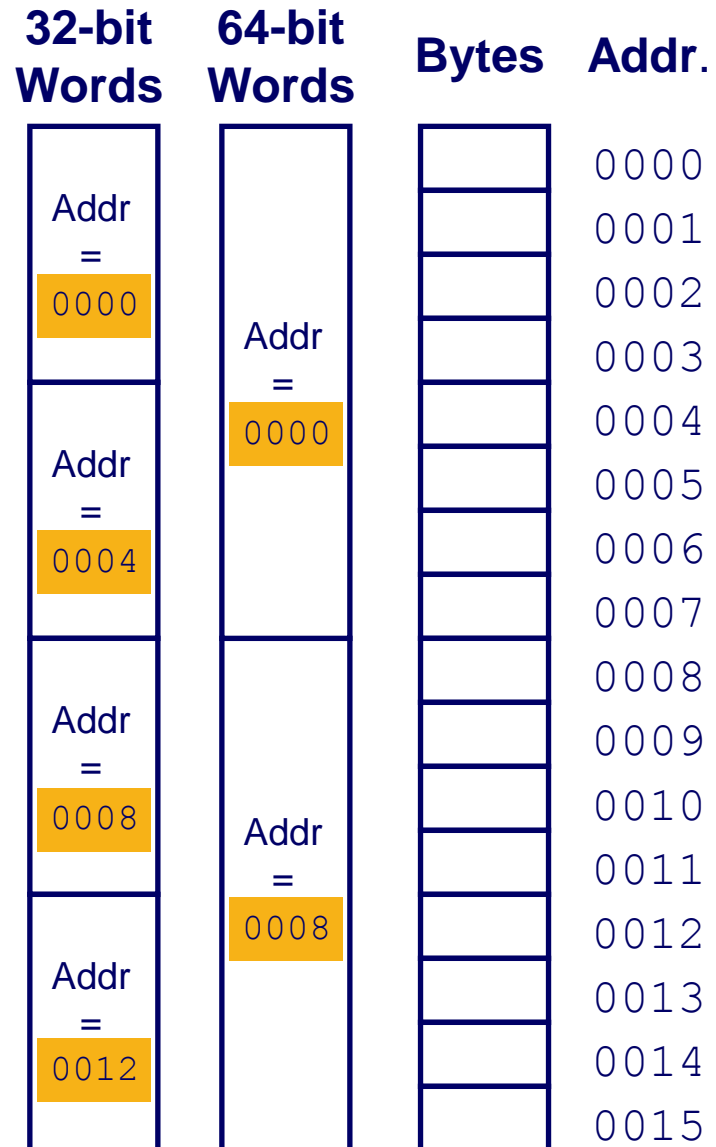
- Programs refer to data by address
 - Conceptually, envision it as a very large array of bytes
 - In reality, it's not, but can think of it that way
 - An address is like an index into that array
 - and, a pointer variable stores an address
- Note: system provides private address spaces to each “process”
 - Think of a process as a program being executed
 - So, a program can clobber its own data, but not that of others



Machine Words

- Word is a native unit of information handled by computer
- Any computer has a “Word Size”
 - Nominal size of integer-valued data
 - and of addresses
 - Until recently, most machines used 32 bits (4 bytes) as word size
 - Limits addresses to 4GB (2^{32} bytes)
 - Increasingly, machines have 64-bit word size
 - Potentially, could have 18 EB (exabytes) of addressable memory
 - That's 18.4×10^{18}
 - Machines still support multiple data formats
 - Fractions or multiples of word size
 - Always integral number of bytes

Word-Oriented Memory Organization



- Addresses Specify Byte Locations
 - Address of first byte in word
 - Addresses of successive words differ by 4 (32-bit) or 8 (64-bit)

Byte Ordering

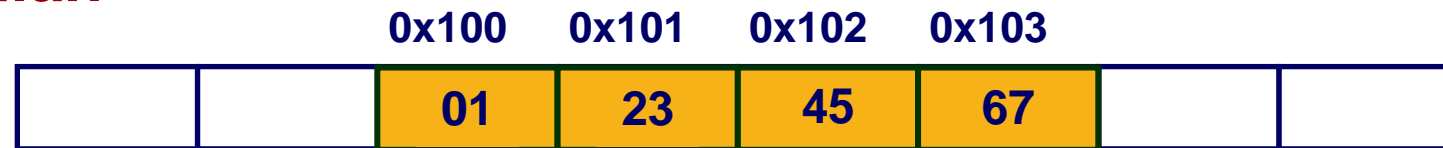
- How are the bytes within a multi-byte word ordered in memory?
- Conventions
 - **Big Endian:** Sun, PPC Mac, Internet
 - Least significant byte has highest address
 - **Little Endian:** x86, ARM processors running Android, iOS, and Windows, RISC-V
 - Least significant byte has lowest address

Byte Ordering Example

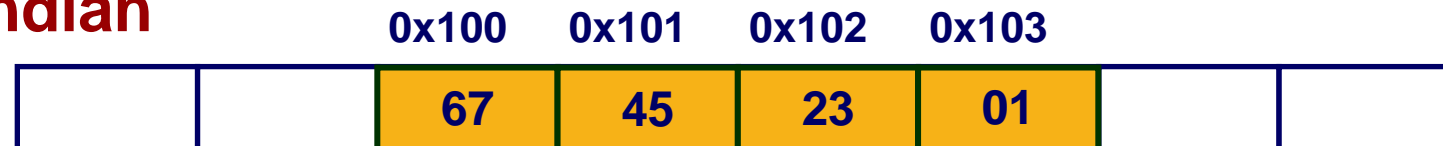
■ Example

- Variable x has 4-byte value of 0x01234567
- Address given by &x is 0x100

Big Endian



Little Endian



Any Questions?

```
                .text
__start:      addi t1, zero, 0x18
                addi t2, zero, 0x21
cycle:        beq t1, t2, done
                slt t0, t1, t2
                bne t0, zero, if_less
                nop
                sub t1, t1, t2
                j cycle
                nop
if_less:      sub t2, t2, t1
                j cycle
done:         add t3, t1, zero
```