CS330 - Computer Organization and Assembly Language Programming

Lecture 3

- Representing and Manipulating Information-

Professor : Mahmut Unan – UAB CS

Agenda

- Binary, Decimal, Hexadecimal
- Information Storage
- Hexadecimal Notation
- Data Sizes
- Addressing and Byte Ordering
- Representing Strings
- Representing Code

- Human Languages; English, Chinese, Greek, Spanish…
  - X = a +b -> x is equal to a plus b
- High Level Programming Languages: Python, Php, R, Matlab…
  - X=a+b;
- Assembly language

```
mov dx, 3c8h
xor al, al
out dx, al
inc dx
mov cx, 256
```

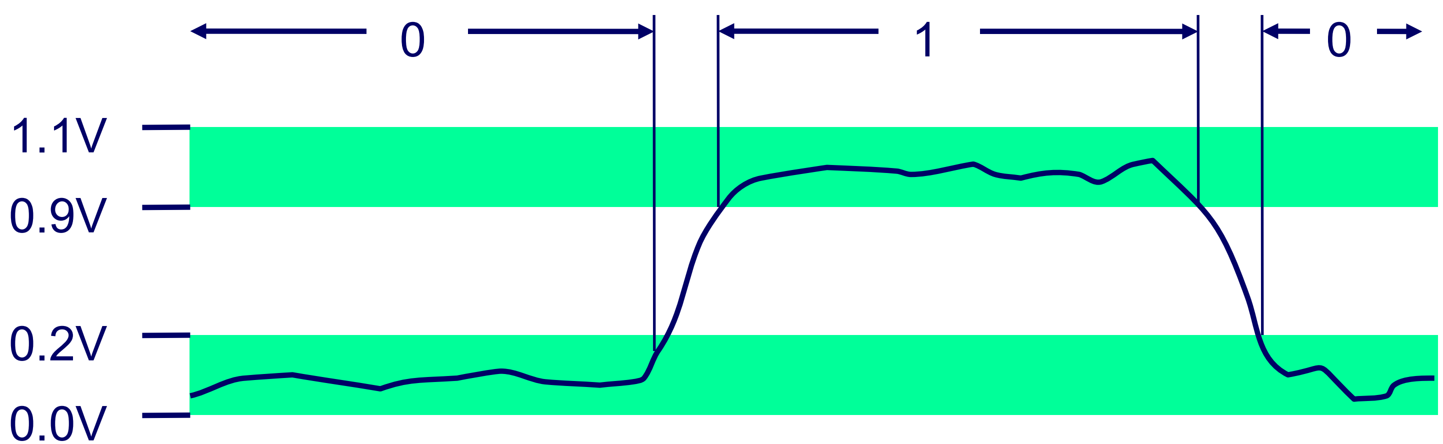
- Strictly speaking, computers can only understand binary instructions: machine language

x = a + b ➔

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

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



For example, can count in binary

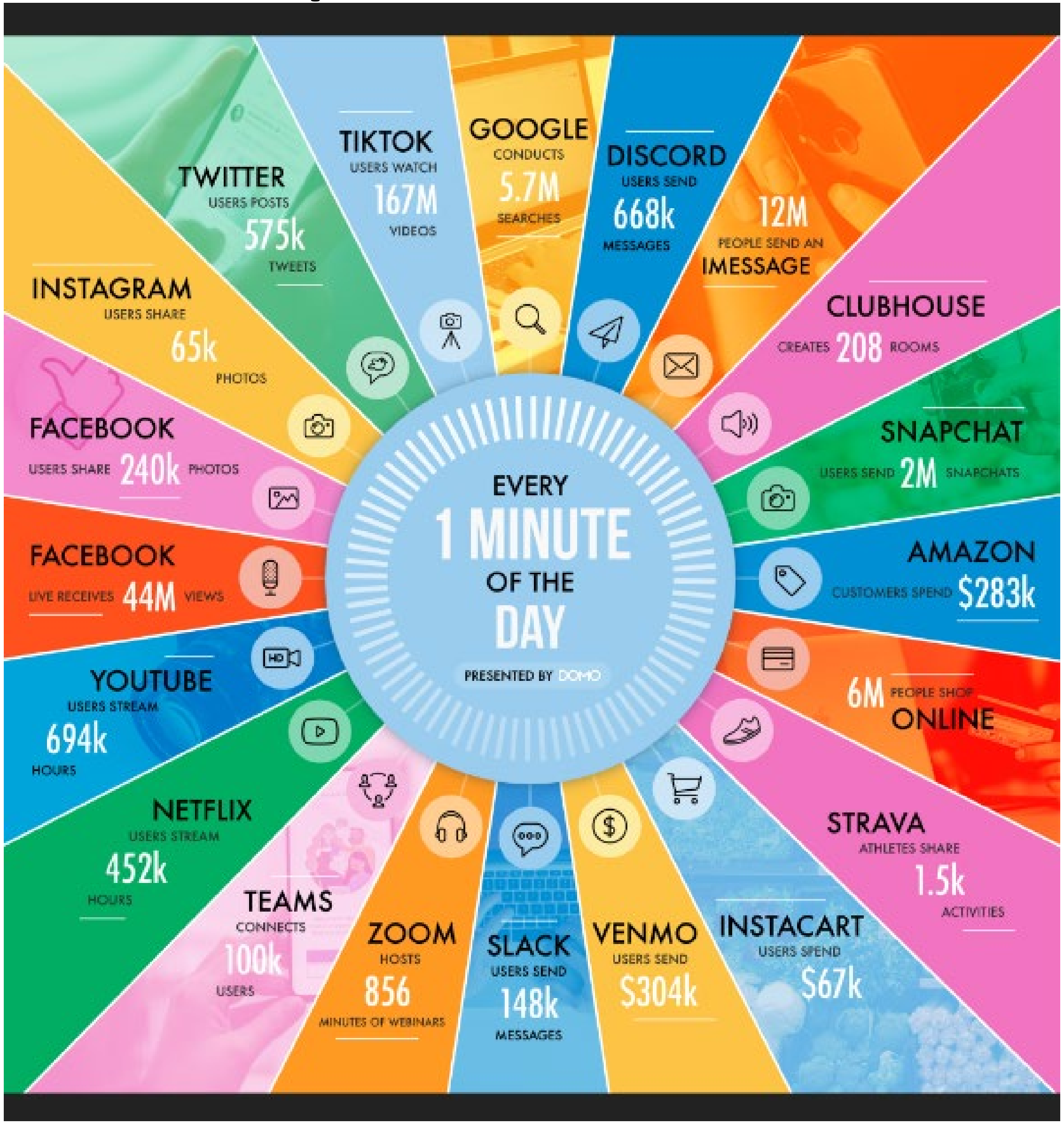
- Base 2 Number Representation
  - Represent 15213_{10} as 11101101101101_2
  - Represent 1.20_{10} as $1.0011001100110011[0011]..._2$
  - Represent 1.5213×10^4 as $1.1101101101101_2 \times 2^{13}$

Bits and Bytes

Data Measurement Chart

Data Measurement	Size
Bit	Single Binary Digit (1 or 0)
Byte	8 bits
Kilobyte (KB)	1,024 Bytes
Megabyte (MB)	1,024 Kilobytes
Gigabyte (GB)	1,024 Megabytes
Terabyte (TB)	1,024 Gigabytes
Petabyte (PB)	1,024 Terabytes
Exabyte (EB)	1,024 Petabytes

Data Never Sleeps



https://www.domo.com/learn/infographic/data-never-sleeps-9

Exercises

- 5 GB = 5*1024*1024*1024 Byte
- 21 TB = …21*1024*1024*1024….. KB
- 76 MB = …76*1024*1024*8…. Bits
- 4 Exabyte= …4*1024*1024*1024… GB
- 2048 GB = …2… TB

Why don't computers use Base 10?

- Base 10 number representation
  - "Digit" in many languages also refers to fingers/toes
    - Of course, decimal (from Latin decimus) , means tenth
  - A position numeral system (unlike, say Roman numerals)
  - Natural representation for financial transactions
  - Even carries through in scientific notation
- Implementing electronically
  - Hard to store
    - ENIAC (First electronic computer)
      - used 10 vacuum tubes / digit
  - Hard to transmit
- • Need high precision to encode 10 signal levels on single wire
  - Messy to implement digital logic functions
    * Addition, multiplication, etc.

http://www.aqualab.cs.northwestern.edu

The Decimal System

- System based on decimal digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) to represent numbers
- For example the number 83 means eight tens plus three:

$$83 = (8 * 10) + 3$$

- The number 4728 means four thousands, seven hundreds, two tens, plus eight:

$$4728 = (4 * 1000) + (7 * 100) + (2 * 10) + 8$$

- The decimal system is said to have a ***base*, or *radix*,** of 10. This means that each digit in the number is multiplied by 10 raised to a power corresponding to that digit's position:

$$83 = (8 * 10^1) + (3 * 10^0)$$

$$4728 = (4 * 10^3) + (7 * 10^2) + (2 * 10^1) + (8 * 10^0)$$

Positional Interpretation of a Decimal Number

4	7	2
100s 10^2 position 2	10s 10^1 position 1	1s 10^0 position 0

$$4*100 + 7*10 + 2*1 = 472$$

Positional Interpretation of a Number in Base 7

Position	4	3	2	1	0
Value in exponential form	7^4	7^3	7^2	7^1	7^0
Decimal value	2401	343	49	7	1

0 0 4 1 5

$(415)_7 = (\quad)_{10}$

$(4*7^2+ 1*7^1+5*7^0) = (196+7+5) =(208)_{10}$

The Binary System

- Only two digits, 1 and 0
- Represented to the base 2
- The digits 1 and 0 in binary notation have the same meaning as in decimal notation:

$$0_2 = 0_{10}$$
$$1_2 = 1_{10}$$

- To represent larger numbers each digit in a binary number has a value depending on its position:

$$10_2 = (1 * 2^1) + (0 * 2^0) = 2_{10}$$
$$11_2 = (1 * 2^1) + (1 * 2^0) = 3_{10}$$
$$100_2 = (1 * 2^2) + (0 * 2^1) + (0 * 2^0) = 4_{10}$$

For representing numbers in base 2, there are two possible digits (0, 1) in which column values are a power of two:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
128	64	32	16	8	4	2	1	
0	**1**	**1**	**0**	**0**	**0**	**1**	**1**	
0 +	64 +	32 +	0 +	0 +	0 +	2 +	1	= **99**

Although values represented in base 2 are significantly longer than those in base 10, **binary representation is used in digital computing because of the resulting simplicity of hardware design**

For representing numbers in base 2, there are two possible digits (0, 1) in which column values are a power of two:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
128	64	32	16	8	4	2	1	
0	**1**	**1**	**0**	**0**	**0**	**1**	**1**	
0 +	64 +	32 +	0 +	0 +	0 +	2 +	1	= **99**
0	1	1	1	1	1	0	0	(011111100)

64+32+16+8+4 =124

For representing numbers in base 2, there are two possible digits (0, 1) in which column values are a power of two:

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

157-128= 29

29-16=13

13-8=5

5-4=1

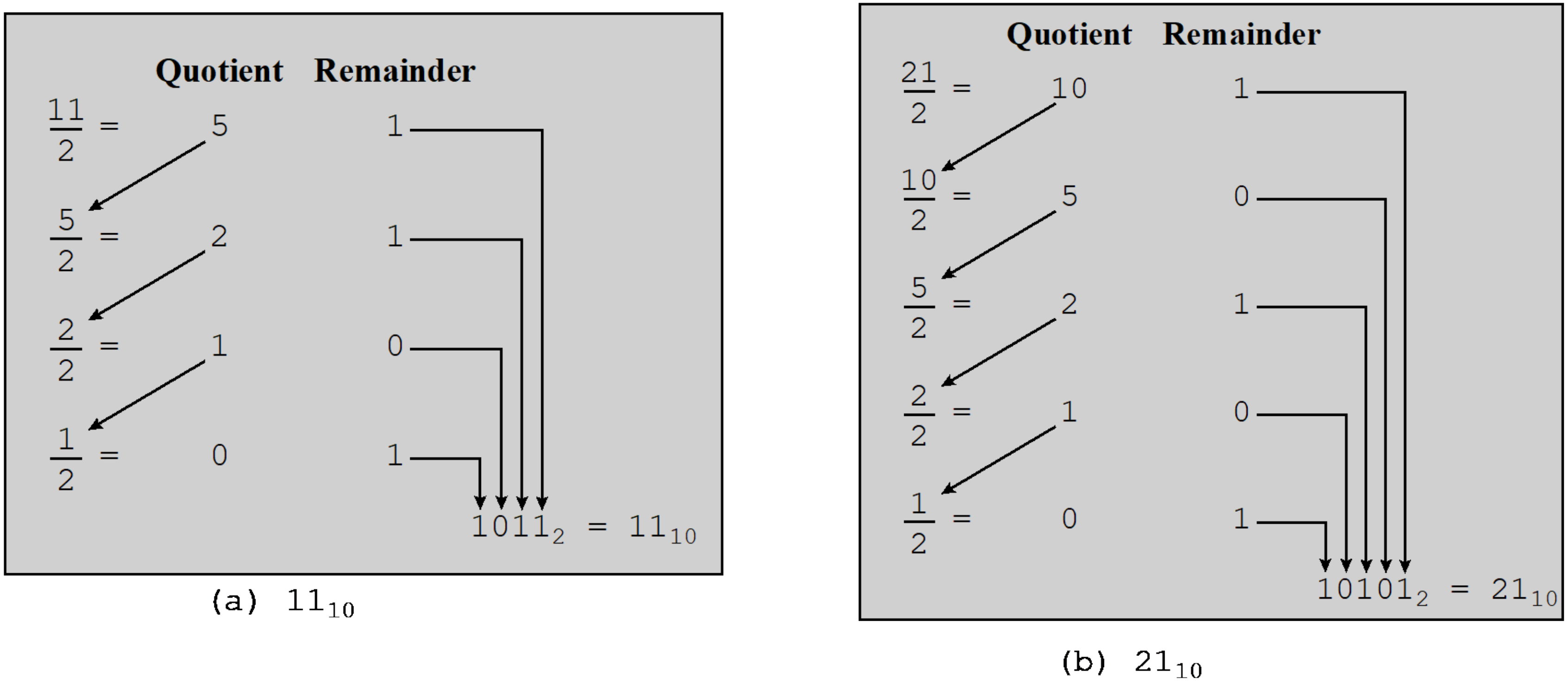


Figure 9.1 Examples of Converting from Decimal Notation to Binary Notation for Integers

Hexadecimal Notation

- Binary digits are grouped into sets of four bits, called a *nibble*
- Each possible combination of four binary digits is given a symbol, as follows:

0000 = 0	0100 = 4	1000 = 8	1100 = C
0001 = 1	0101 = 5	1001 = 9	1101 = D
0010 = 2	0110 = 6	1010 = A	1110 = E
0011 = 3	0111 = 7	1011 = B	1111 = F

- Because 16 symbols are used, the notation is called *hexadecima*l and the 16 symbols are the *hexadecimal digits*
- Thus

$$2C_{16} = (2_{16} * 16^1) + (C_{16} * 16^0)$$
$$= (2_{10} * 16^1) + (12_{10} * 16^0) = 44 = 0010\ 1100$$

Hexadecimal Notation

- Binary digits are grouped into sets of four bits, called a *nibble*
- Each possible combination of four binary digits is given a symbol, as follows:

0000 = 0	0100 = 4	1000 = 8	1100 = C
0001 = 1	0101 = 5	1001 = 9	1101 = D
0010 = 2	0110 = 6	1010 = A	1110 = E
0011 = 3	0111 = 7	1011 = B	1111 = F

F3360D

1111 0011 0011 0110 0000 1101

Decimal, Binary, and Hexadecimal

Decimal (base 10)	Binary (base 2)	Hexadecimal (base 16)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	0001 0000	10
17	0001 0001	11
18	0001 0010	12
31	0001 1111	1F
100	0110 0100	64
255	1111 1111	FF
256	0001 0000 0000	100

Byte-Oriented Memory Organization

- Programs refer to virtual addresses
  - Conceptually very large array of bytes
  - Each byte with its own address
  - All addresses – virtual address space
  - In Unix and Windows, address space private to particular “process”
    - Program being executed
    - Program can manipulate its own data, but not that of others
- Compiler + run-time system control allocation
  - Where different program objects should be stored
  - Multiple mechanisms: static, stack, and heap
  - In any case, all allocation within single virtual address space

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

Hex	Decimal	Binary
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

1100 1001 0111 1011 → 0xC97B

Machine Words

- Machine has "word size"
  - Nominal size of integer-valued data
  - More importantly – a virtual address is encoded by such a word
    - Hence, it determines max size of virtual address space
  - Most current machines are 32 bits (4 bytes)
    - Limits addresses to 4GB
    - Becoming too small for memory-intensive applications
  - Newer systems are 64 bits (8 bytes)
    - Potentially address ≈ 1.84 X 10^19 bytes
  - Machines support multiple data formats
    - Fractions or multiples of word size
    - Always integral number of bytes

Data Sizes

- Each computer has a word size
  - For a machine with w-bit word size
  - The virtual address can range from 0 to 2^w-1
  - The program access to at most 2^w bytes

- 32 bit vs 64 bit

Example Data Representations

C Data Type	Typical 32-bit	Typical 64-bit	x86-64
`char`	1	1	1
`short`	2	2	2
`int`	4	4	4
`long`	4	8	8
`float`	4	4	4
`double`	8	8	8
`long double`	−	−	10/16
pointer	4	8	8

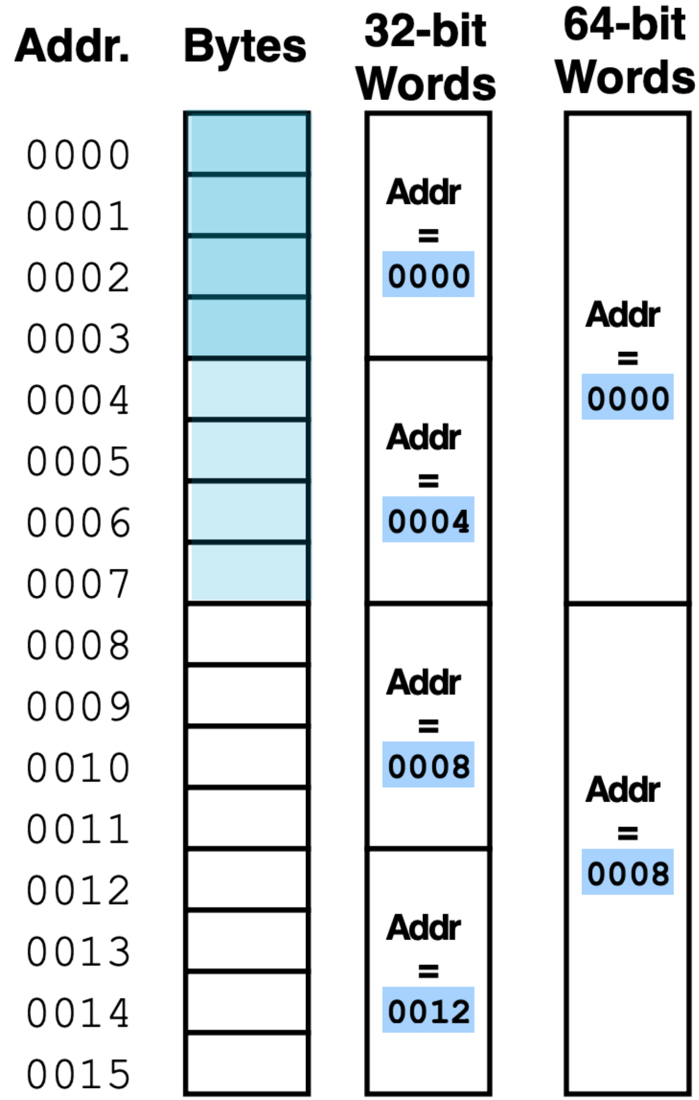
Addressing and Byte Ordering

- For objects that span multiple bytes (e.g. integers), we need to agree on two things
  - what would be the address of the object?
  - how would we order the bytes in memory?

http://www.aqualab.cs.northwestern.edu/component/attachments/download/167

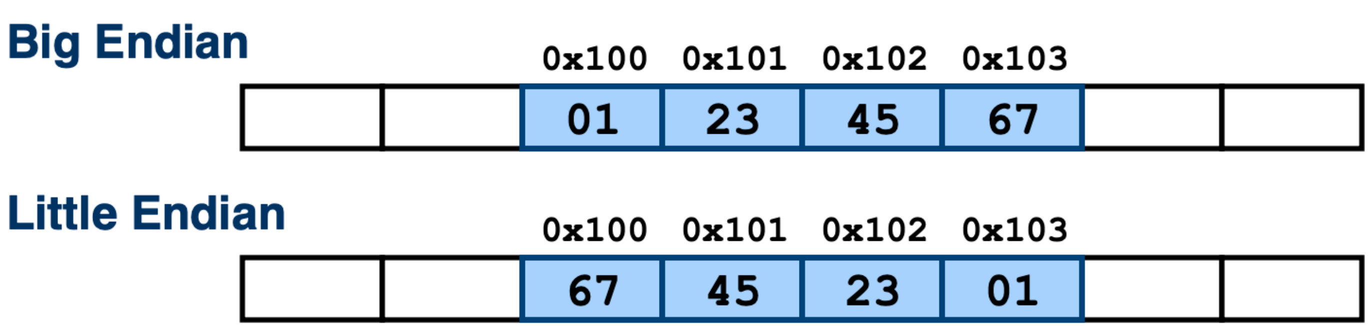
Word-Oriented Memory Organization

- 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 to order bytes within multi-byte word in memory
- Conventions
  - (most) Sun’s, IBMs are “Big Endian” machines
    - Least significant byte has highest address (comes last)
  - (most) Intel’s are “Little Endian” machines
    - Least significant byte has lowest address (comes first)
- Example
  - Variable x has 4-byte representation 0x01234567
  - Address given by &x is 0x100 0x100 0x101



http://www.aqualab.cs.northwestern.edu/component/attachments/download/167

Examining Data Representations

- Code to print byte representation of data
  - Casting pointer to `unsigned char *` creates byte array

```
typedef unsigned char *pointer;
void show_bytes(pointer start, int len)
{
 int i;
for (i = 0; i < len; i++) {
printf("0x%p\t0x%.2x\n", start+i, start[i]);
printf("\n");
}
}
```

Printf directives:
%p: Print pointer
%x: Print Hexadecimal

http://www.aqualab.cs.northwestern.edu/component/attachments/download/167

`show_bytes` Execution Example

```
int a = 15213;
printf("int a = 15213;\n");
show_bytes((pointer) &a, sizeof(int));
```

Result (Linux):

```
int a = 15213;
0x11ffffcb8 0x6d
0x11ffffcb9 0x3b
0x11ffffcba 0x00
0x11ffffcbb 0x00
```

$0011\ 1011\ 0110\ 1101_2$

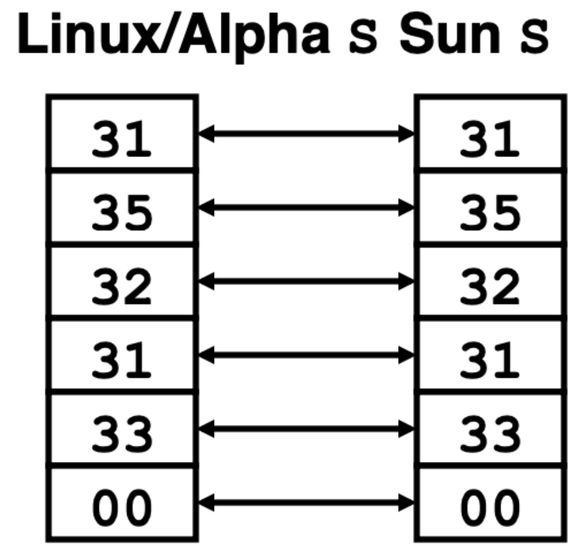
$3 \quad b \quad 6 \quad d_{16}$

http://www.aqualab.cs.northwestern.edu/component/attachments/download/167

Representing Strings

- Strings in C
  - Represented by array of characters
  - Each character encoded in ASCII format
    - Standard 7-bit encoding of character set
    - Other encodings exist, but uncommon
    - Character “0” has code 0x30
      - » Digit i has code 0x30+i
  - String should be null-terminated
    - Final character = 0
- Compatibility
  - Byte ordering not an issue
    - Data are single byte quantities
  - Text files generally platform independent
    - Except for different conventions of line termination character(s)!

```
char S[6] = "15213";
```



http://www.aqualab.cs.northwestern.edu/component/attachments/download/167

Machine-level Code Representation

- Encode program as sequence of instructions
  - Each simple operation
    - Arithmetic operation
    - Read or write memory
    - Conditional branch
  - Instructions encoded as bytes
    - Alpha's, Sun's, Mac's use 4 byte instructions
      - » Reduced Instruction Set Computer (RISC)
    - PC's use variable length instructions
      - » Complex Instruction Set Computer (CISC)
  - Different machines → different ISA & encodings
    - Most code not binary compatible
- A fundamental concept:
  - Programs are byte sequences too!

Representing Instructions

```
int sum(int x, int y) {
return x+y;
}
```

- Sun use 2 4-byte instructions
  - Differing numbers in other cases
- PC uses instructions with lengths 1, 2, and 3 bytes
  - Mostly the same for NT and for Linux
  - NT / Linux not fully binary compatible

Linux 32	55	89	E5	8B	45	0C	03	45	08	C9	C3
Windows	55	89	E5	8B	45	0C	03	45	08	5D	C3
Sun	81	C3	E0	08	90	02	00	09			

Different machines use totally different instructions and encodings

http://www.aqualab.cs.northwestern.edu/component/attachments/download/167

Next Class

- We will continue **Chapter 2**
- **Boolean Algebra**
- Please read **2.1.6 – 2.1.9**