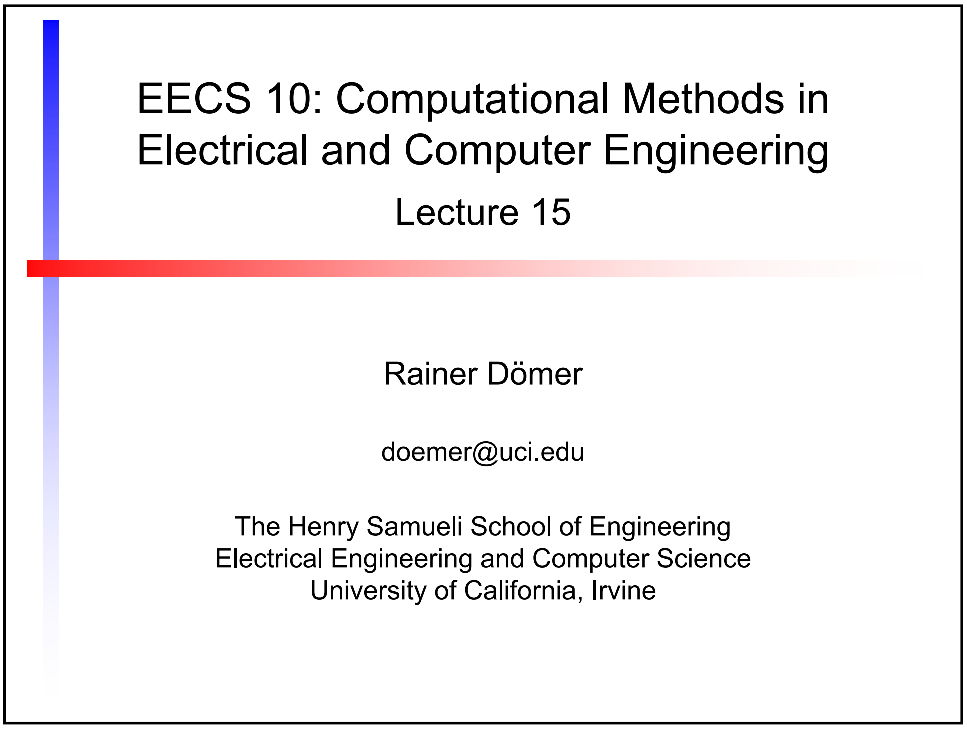


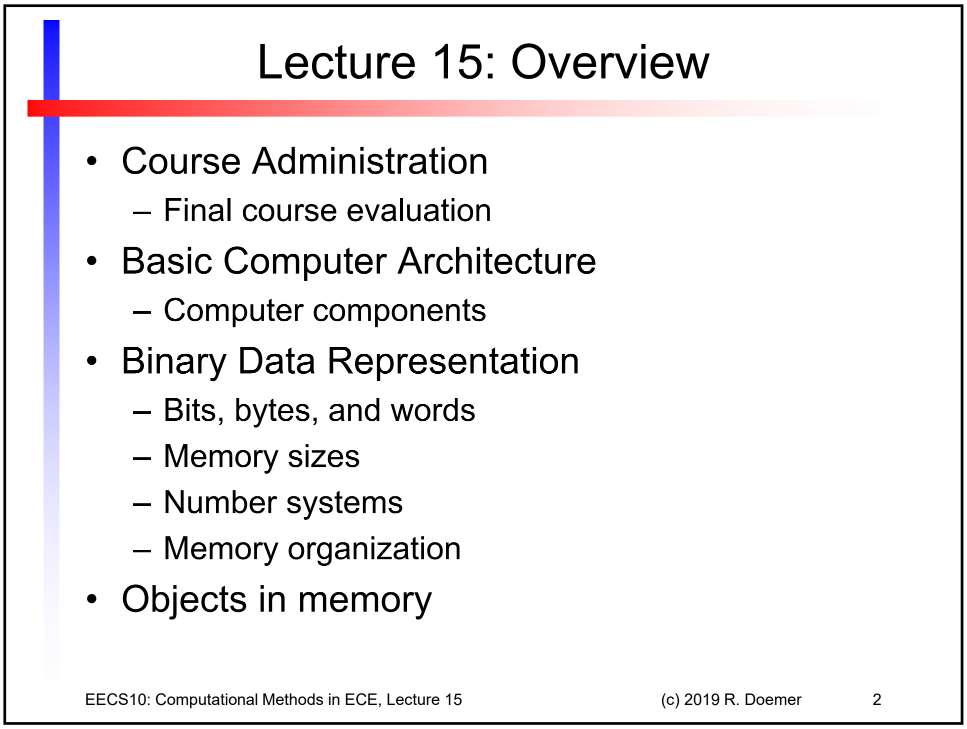
EECS 10: Computational Methods in Electrical and Computer Engineering

Lecture 15

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine



Lecture 15: Overview

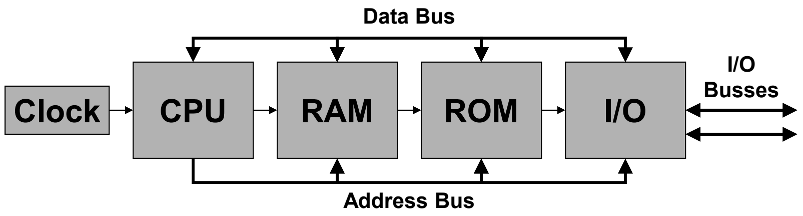
- Course Administration
  - Final course evaluation
- Basic Computer Architecture
  - Computer components
- Binary Data Representation
  - Bits, bytes, and words
  - Memory sizes
  - Number systems
  - Memory organization
- Objects in memory

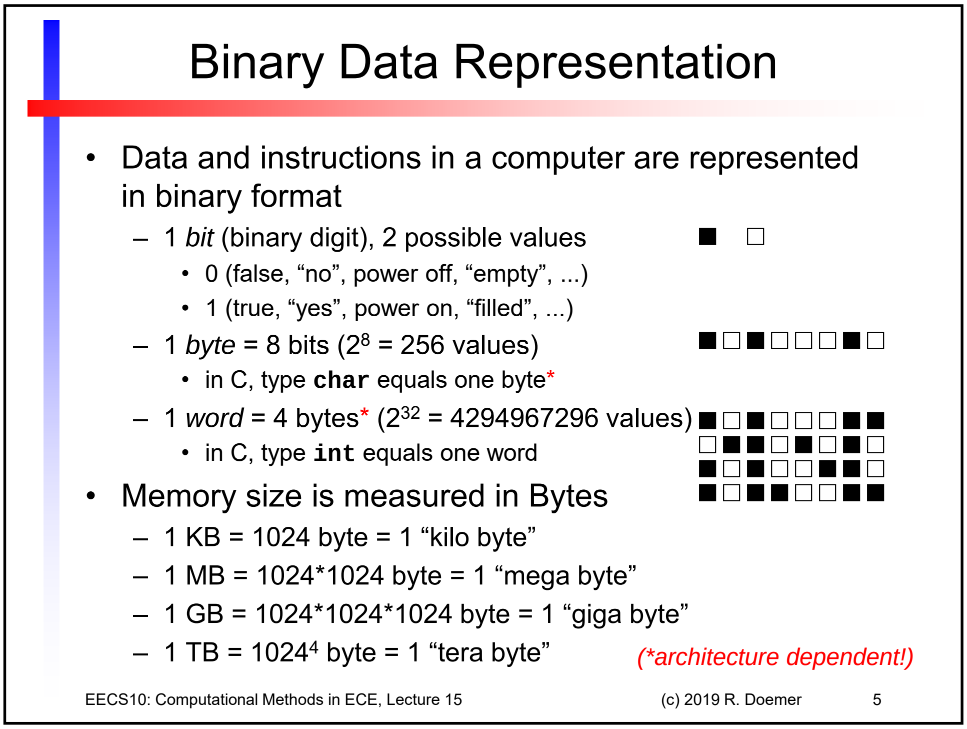
Course Administration

- Final Course Evaluation
  - Open three weeks
  - Nov. 21, 2019, through Sunday, Dec. 8, 2019
  - Online via EEE Evaluation application
- Mandatory Evaluation of Course and Instructor
  - Voluntary
  - Anonymous
  - Very valuable
    - Help to improve this class!
- Please spend 5 minutes!

Basic Computer Architecture

- Essential Computer Components
  - Central Processing Unit (CPU)
    - e.g. Intel Pentium, Motorola PowerPC, Sun SPARC, ...
  - Random Access Memory (RAM)
    - storage for program and data, read and write access
  - Read Only Memory (ROM)
    - fixed storage for basic input/output system (BIOS)
  - I/O Units
    - Input/output interfaces connecting to peripherals





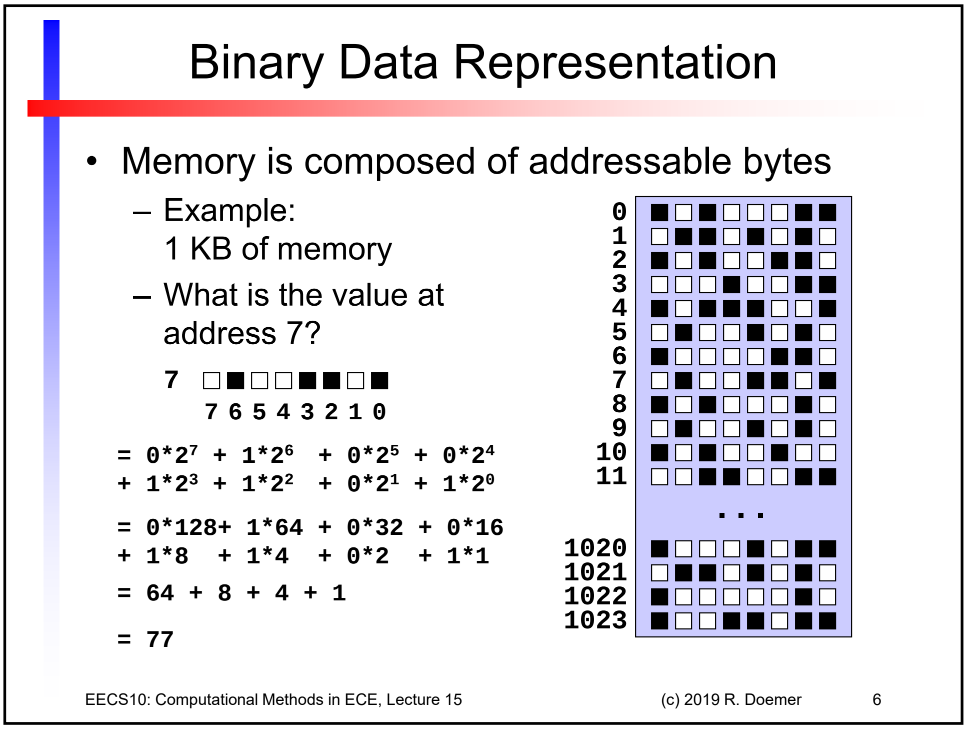
Binary Data Representation

- Data and instructions in a computer are represented in binary format
  - 1 *bit* (binary digit), 2 possible values ■ □
    - 0 (false, “no”, power off, “empty”, ...)
    - 1 (true, “yes”, power on, “filled”, ...)
  - 1 *byte* = 8 bits (2^8 = 256 values) ■□■□□□■□
    - in C, type **`char`** equals one byte*
  - 1 *word* = 4 bytes* (2^{32} = 4294967296 values)
    - in C, type **`int`** equals one word

■□■□□□■■
□■■□■□■□
■□■□□■■□
■□■■□□■■

- Memory size is measured in Bytes
  - 1 KB = 1024 byte = 1 “kilo byte”
  - 1 MB = 1024*1024 byte = 1 “mega byte”
  - 1 GB = 1024*1024*1024 byte = 1 “giga byte”
  - 1 TB = 1024^4 byte = 1 “tera byte”

*(*architecture dependent!)*



Binary Data Representation

- Memory is composed of addressable bytes
  - Example:
    1 KB of memory
  - What is the value at address 7?

```
7  □■□□■■□■
    7 6 5 4 3 2 1 0
```

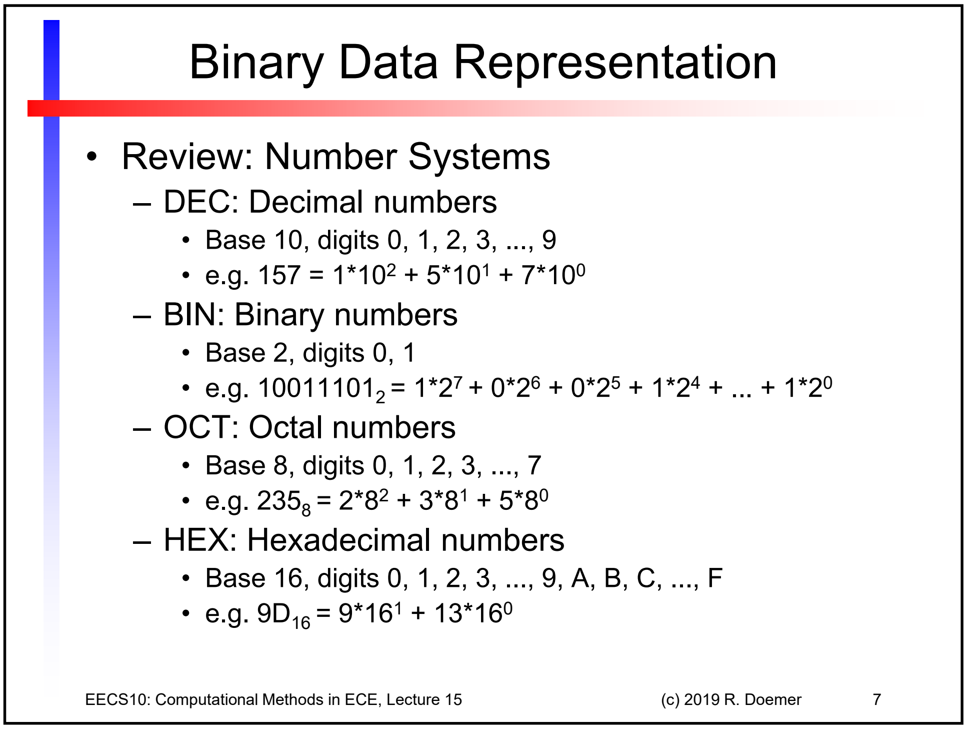
$$= 0*2^7 + 1*2^6 + 0*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0$$

$$= 0*128 + 1*64 + 0*32 + 0*16 + 1*8 + 1*4 + 0*2 + 1*1$$

$$= 64 + 8 + 4 + 1$$

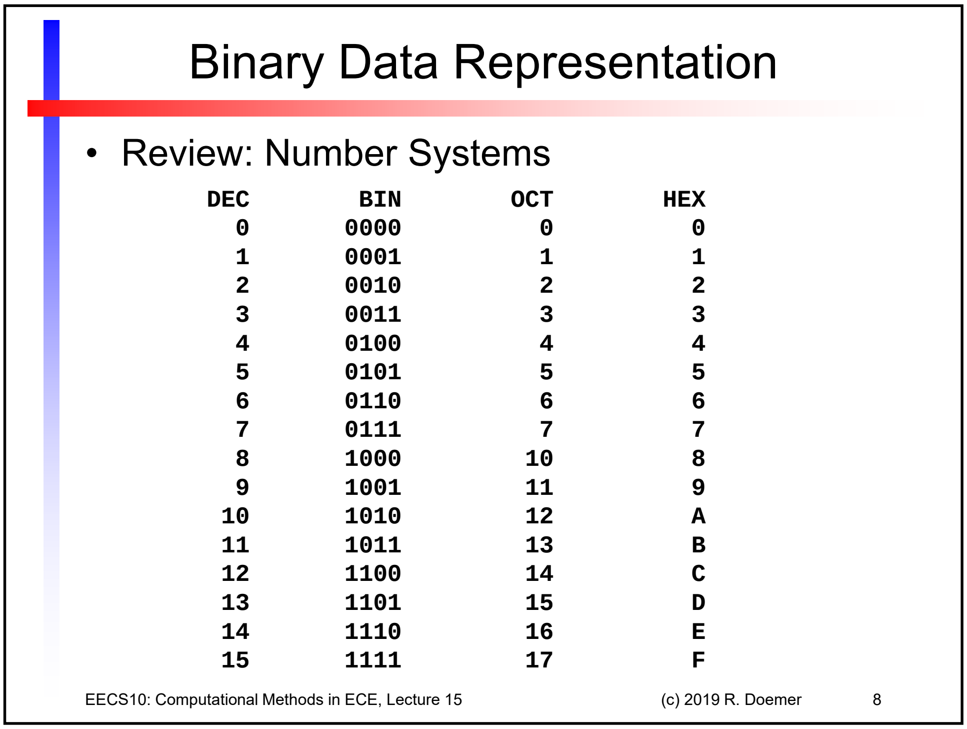
$$= 77$$

0	■□■□□□■■
1	□■■□■□■□
2	■□■□□■■□
3	□□□■□□■■
4	■□■■■□□■
5	□■□□■□■□
6	■□□□□■■□
7	□■□□■■□■
8	■□■□□□■□
9	□■□□■□■□
10	■□■□□■□□
11	□□■■□□■■
	...
1020	■□□□■□■■
1021	□■■□■□■□
1022	■□□□□□■□
1023	■□□■■□■■



Binary Data Representation

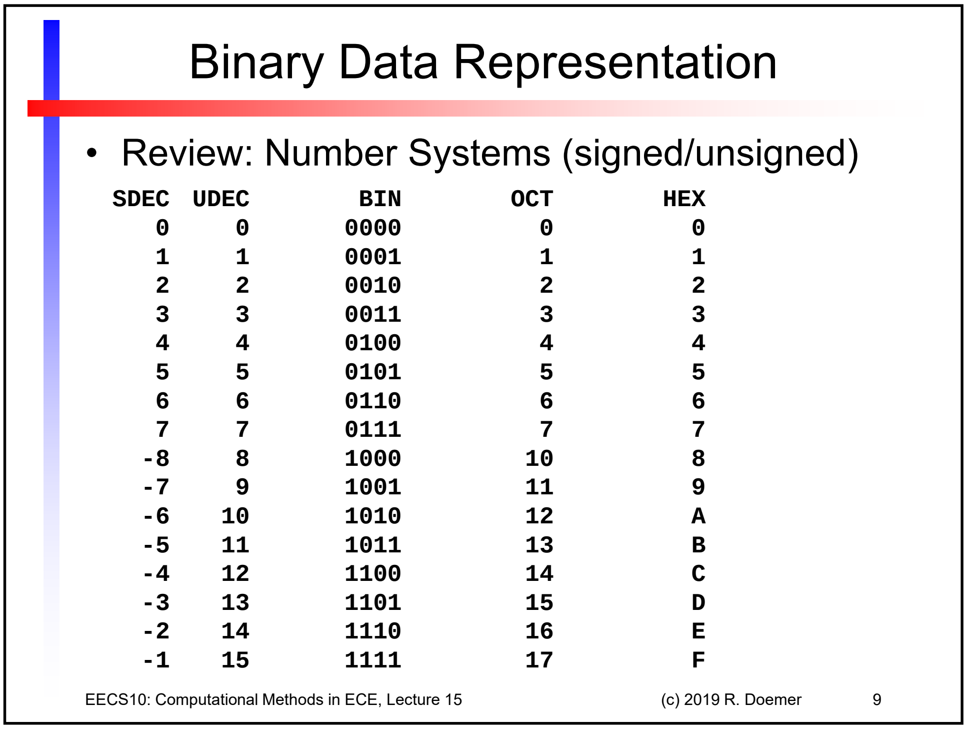
- Review: Number Systems
  - DEC: Decimal numbers
    - Base 10, digits 0, 1, 2, 3, ..., 9
    - e.g. $157 = 1*10^2 + 5*10^1 + 7*10^0$
  - BIN: Binary numbers
    - Base 2, digits 0, 1
    - e.g. $10011101_2 = 1*2^7 + 0*2^6 + 0*2^5 + 1*2^4 + ... + 1*2^0$
  - OCT: Octal numbers
    - Base 8, digits 0, 1, 2, 3, ..., 7
    - e.g. $235_8 = 2*8^2 + 3*8^1 + 5*8^0$
  - HEX: Hexadecimal numbers
    - Base 16, digits 0, 1, 2, 3, ..., 9, A, B, C, ..., F
    - e.g. $9D_{16} = 9*16^1 + 13*16^0$



Binary Data Representation

- Review: Number Systems

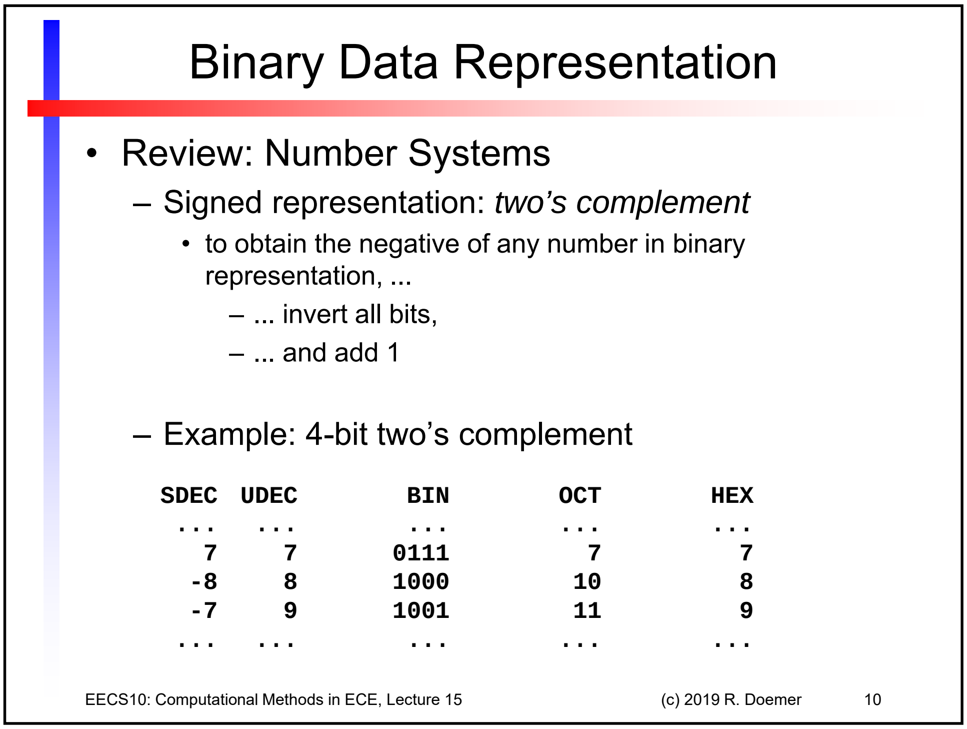
DEC	BIN	OCT	HEX
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F



Binary Data Representation

- Review: Number Systems (signed/unsigned)

SDEC	UDEC	BIN	OCT	HEX
0	0	0000	0	0
1	1	0001	1	1
2	2	0010	2	2
3	3	0011	3	3
4	4	0100	4	4
5	5	0101	5	5
6	6	0110	6	6
7	7	0111	7	7
-8	8	1000	10	8
-7	9	1001	11	9
-6	10	1010	12	A
-5	11	1011	13	B
-4	12	1100	14	C
-3	13	1101	15	D
-2	14	1110	16	E
-1	15	1111	17	F



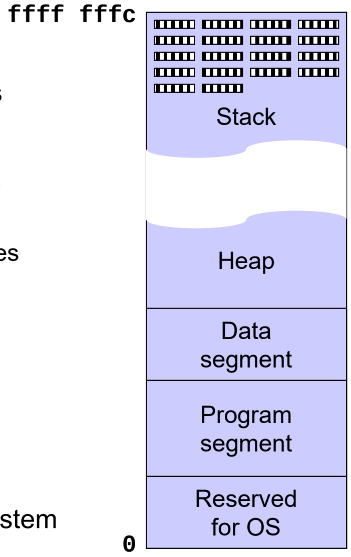
Binary Data Representation

- Review: Number Systems
  - Signed representation: *two's complement*
    - to obtain the negative of any number in binary representation, ...
      - ... invert all bits,
      - ... and add 1
  - Example: 4-bit two's complement

SDEC	UDEC	BIN	OCT	HEX
...	...	...	...	...
7	7	0111	7	7
-8	8	1000	10	8
-7	9	1001	11	9
...	...	...	...	...

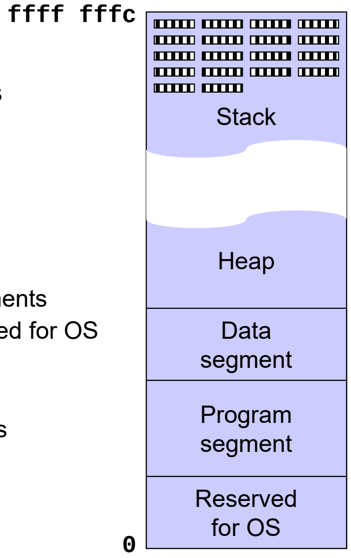
Memory Organization

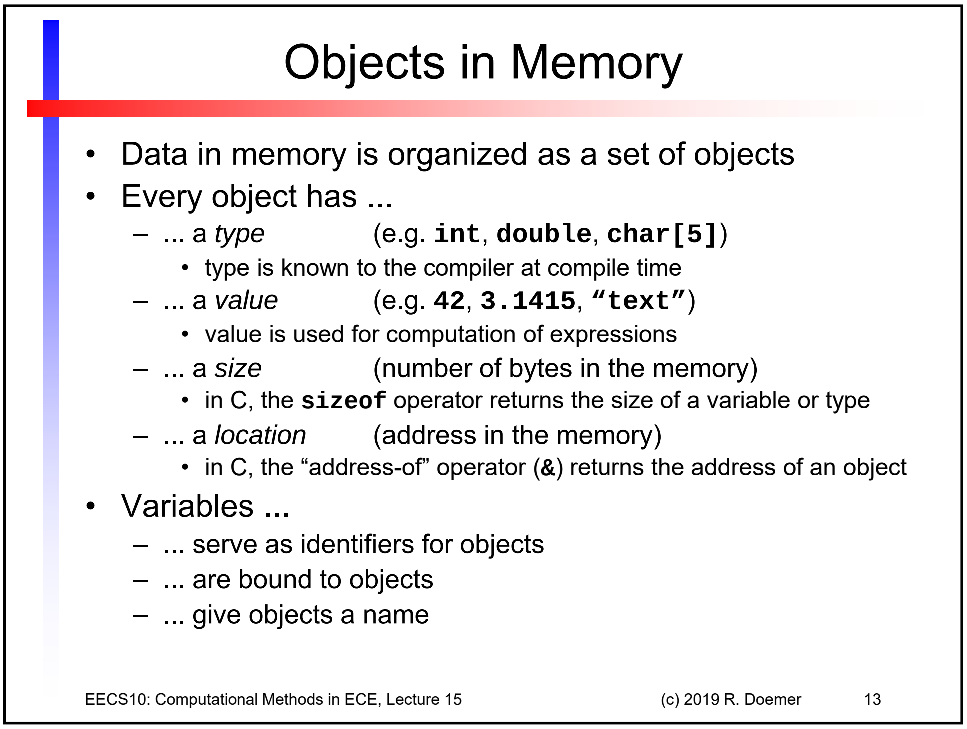
- Memory Segmentation
  - typical (virtual) memory layout on processor with 4-byte words and 4 GB of memory
  - Stack
    - grows and shrinks dynamically
    - function call hierarchy
    - stack frames with local variables
  - Heap
    - “free” storage
    - dynamic allocation by the user
  - Data segment
    - global (and static) variables
  - Program segment
    - stores binary program code
  - Reserved area for operating system



Memory Organization

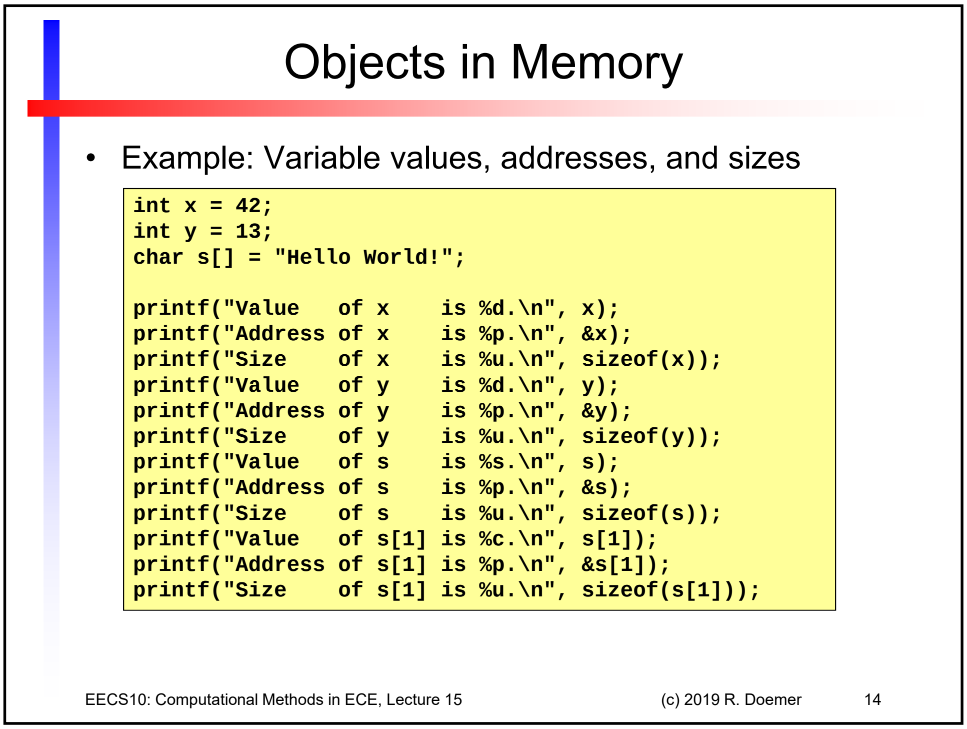
- Memory Segmentation
  - typical (virtual) memory layout on processor with 4-byte words and 4 GB of memory
- Memory errors
  - *Out of memory*
    - Stack and heap collide
  - *Segmentation fault*
    - access outside allocated segments
    - e.g. access to segment reserved for OS
  - *Bus error*
    - mis-aligned word access
    - e.g. word access to an address that is not divisible by 4





Objects in Memory

- Data in memory is organized as a set of objects
- Every object has ...
  - ... a *type* (e.g. **`int`**, **`double`**, **`char[5]`**)
    - type is known to the compiler at compile time
  - ... a *value* (e.g. **`42`**, **`3.1415`**, **`“text”`**)
    - value is used for computation of expressions
  - ... a *size* (number of bytes in the memory)
    - in C, the **`sizeof`** operator returns the size of a variable or type
  - ... a *location* (address in the memory)
    - in C, the “address-of” operator (**&**) returns the address of an object
- Variables ...
  - ... serve as identifiers for objects
  - ... are bound to objects
  - ... give objects a name

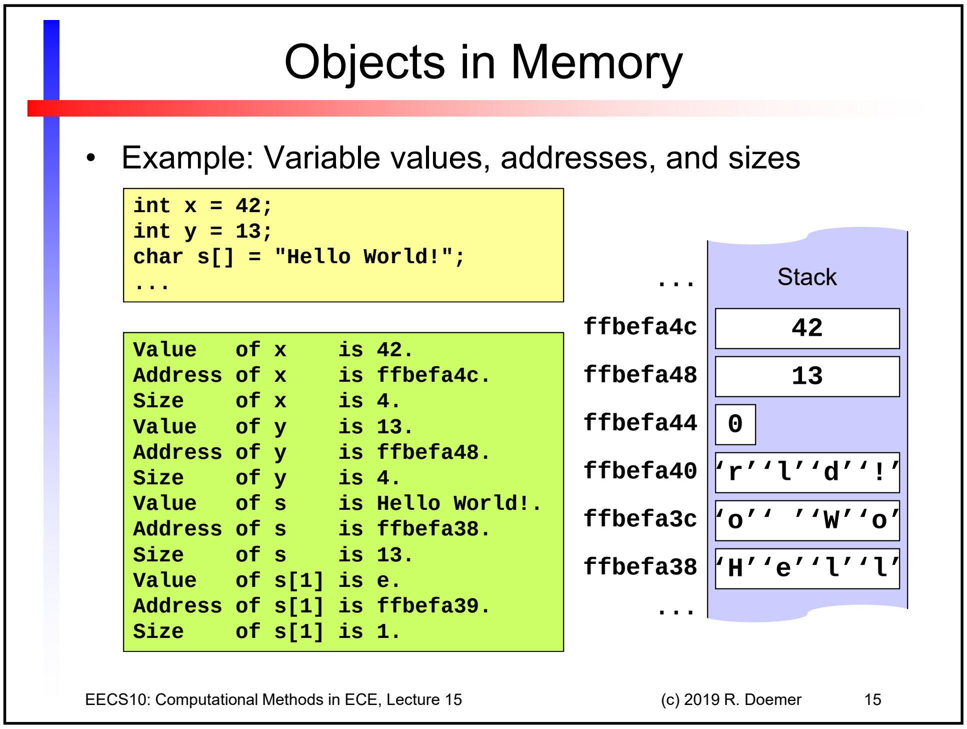


Objects in Memory

- Example: Variable values, addresses, and sizes

```
int x = 42;
int y = 13;
char s[] = "Hello World!";

printf("Value   of x    is %d.\n", x);
printf("Address of x    is %p.\n", &x);
printf("Size    of x    is %u.\n", sizeof(x));
printf("Value   of y    is %d.\n", y);
printf("Address of y    is %p.\n", &y);
printf("Size    of y    is %u.\n", sizeof(y));
printf("Value   of s    is %s.\n", s);
printf("Address of s    is %p.\n", &s);
printf("Size    of s    is %u.\n", sizeof(s));
printf("Value   of s[1] is %c.\n", s[1]);
printf("Address of s[1] is %p.\n", &s[1]);
printf("Size    of s[1] is %u.\n", sizeof(s[1]));
```



Objects in Memory

- Example: Variable values, addresses, and sizes

```
int x = 42;
int y = 13;
char s[] = "Hello World!";
...
```

```
Value   of x    is 42.
Address of x    is ffbefa4c.
Size    of x    is 4.
Value   of y    is 13.
Address of y    is ffbefa48.
Size    of y    is 4.
Value   of s    is Hello World!.
Address of s    is ffbefa38.
Size    of s    is 13.
Value   of s[1] is e.
Address of s[1] is ffbefa39.
Size    of s[1] is 1.
```

	Stack
...	
ffbefa4c	42
ffbefa48	13
ffbefa44	0
ffbefa40	‘r’‘l’‘d’‘!’
ffbefa3c	‘o’‘ ’‘W’‘o’
ffbefa38	‘H’‘e’‘l’‘l’
...	