Unit 1

Digital Representation

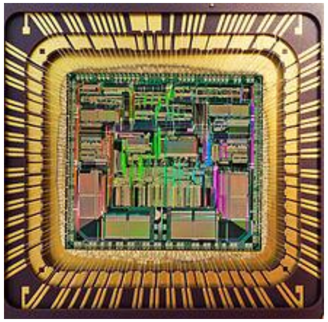
Information Representation

- All information is represented as sequences of 1's and 0's
- Kinds of information
  - Numbers
  - Text
  - Instructions
  - Sound
  - Image/Video

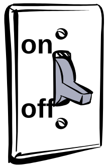
Main Point: All of these forms of information are represented as numbers in a computer and manipulated as such.

Why 1's and 0's

- Modern computer chips are made from millions of tiny transistors built on a chip of silicon (usually)
- A transistor is an electronic device that acts like a switch
  - It can be **on** or **off**
  - This leads to only 2 values (high or low voltage) in computer hardware
  - 1's and 0's are arbitrary symbols representing high or low voltage
- A single 1 or 0 is known as a **bit**



Transistor Acts as a Switch



High Voltage → **1**
+5V
+12V

or

Low Voltage → **0**
0V
-12V

Starting With Numbers

- A single **bit** can only represent 1 and 0
- To represent more than just 2 values we need to use combinations/sequences of many bits
  - A **byte** is defined as a group 8-bits
  - A **word** varies in size but is usually 32-bits

0 or 1

A bit

Two bit Combinations

01000001

An example byte

0101110 11010001 10110101 01110111

An example word

Finite Size

- When we humans solve arithmetic problems, we can just write as many digits as we want
  - If we limited our numbers to 3 digits, our range would be limited to: __________
- Computers store bits in fixed-size units (8-bits, 16-bits, 32-bits, etc.)
  - This limits the range of numbers we can generate and store
- Given **n-bits**, we can make **2^n values**

1	0	1	1	0	0	1	0

Unique Numbers

- Computers represent binary numbers using a fixed number of bits
- Given a fixed number of bits, *n*, what is the range of numbers we can make?

If n=1 bit:

1	
0	= 0
1	= 1

$2^1 = 2$ values

If n=2 bits:

2	1	
0	0	= 0
0	1	= 1
1	0	= 2
1	1	= 3

$2^2 = 4$ values

If n=3 bits:

4	2	1	
0	0	0	= 0
0	0	1	= 1
0	1	0	= 2
0	1	1	= 3
1	0	0	= 4
1	0	1	= 5
1	1	0	= 6
1	1	1	= 7

$2^3 = 8$ values

Given n bits, 2^n numbers can be made

Interpreting Binary Strings

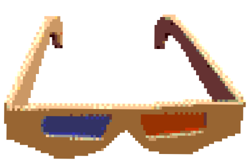
- Given a string of 1's and 0's, you need to know the *representation system* being used, before you can understand the value of those 1's and 0's.
- Information (value) = Bits + Context (System)

01000001 = ?

Unsigned Binary system

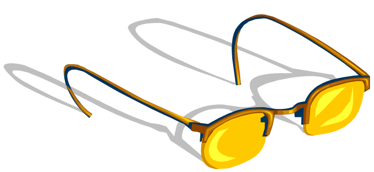
65_{10}

Instructions



"add 1"

ASCII system



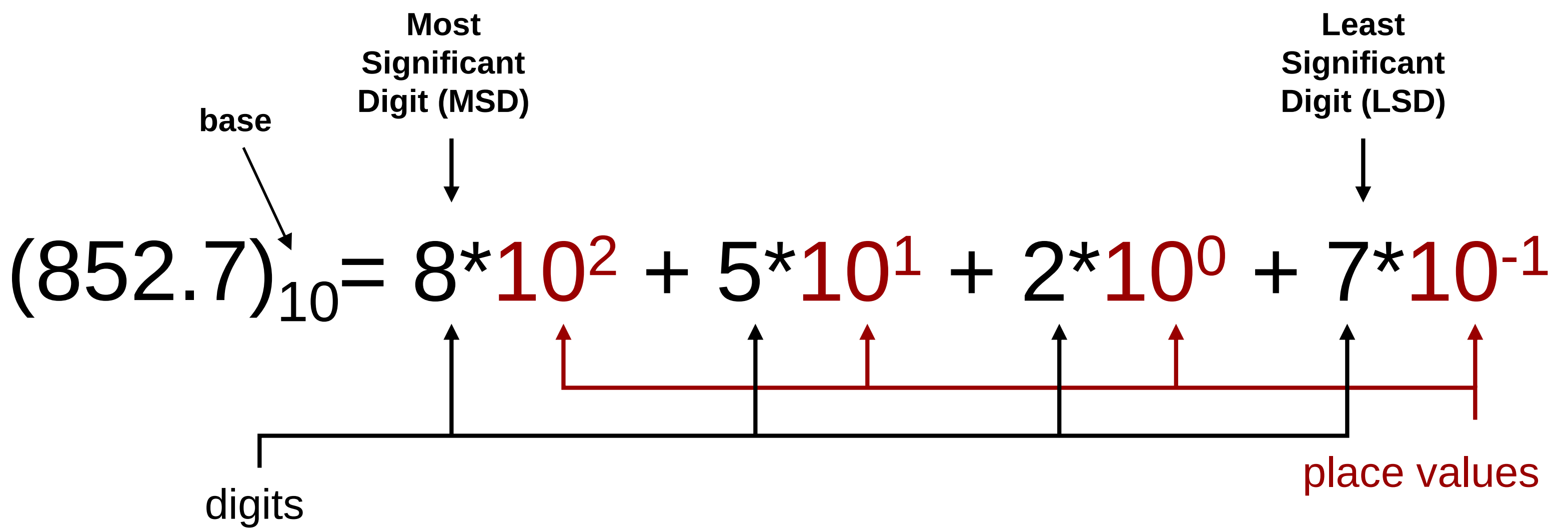
'A'_{ASCII}

Unsigned Binary Number System

- Humans use the decimal number system
  - Based on number 10
  - 10 digits: [0-9]
- Because computer hardware uses digital signals with 2 values, computers use the binary number system
  - Based on number 2
  - 2 binary digits (a.k.a bits): [0,1]

Number System Theory

- The written digits have implied place values
- Place values are powers of the base (decimal = 10)
- Place value of digit to left of decimal point is 10^0 and ascend from there, negative powers of 10 to the right of the decimal point
- The value of the number is the sum of each digit times its implied place value



$$(852.7)_{10} = 8*10^2 + 5*10^1 + 2*10^0 + 7*10^{-1}$$

Binary Number System

- Place values are powers of 2
- The value of the number is the sum of each bit times its implied place value (power of 2)

base

$(110.1)_2 =$

Binary Number System

- Place values are powers of 2
- The value of the number is the sum of each bit times its implied place value (power of 2)

base

Most Significant Bit (MSB)

Least Significant Bit (LSB)

$$(110.1)_2 = 1 * 2^2 + 1 * 2^1 + 0 * 2^0 + 1 * 2^{-1}$$

bits

place values

$$(110.1)_2 = 1*4 + 1*2 + 1*.5 = 4+2+.5 = 6.5_{10}$$

Powers of 2

- Might help to memorize the powers of 2

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

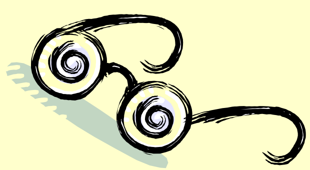
___	___	___	___	___	___	___	___	___	___	___
1024	512	256	128	64	32	16	8	4	2	1

Interpreting Binary Strings

- Information (65) = Bits + Context (Unsigned Binary System)

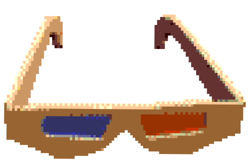
01000001 = ?

Unsigned
Binary system



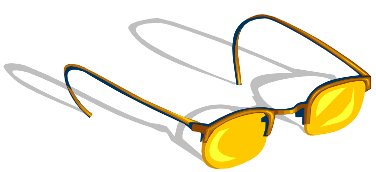
65_{10}

Instructions



"add 1"

ASCII
system



$'A'_{ASCII}$

Sign

- Is there any limitation if we only use the powers of some base as our weights?
  - Can't make negative numbers
- What if we change things
  - How do humans represent negative numbers?
  - Can we do something similar?

___	___	___	___	___	___	___	___
	64	32	16	8	4	2	1

___	___	___	___	___	___	___	___
128	64	32	16	8	4	2	1

C Integer Data Types

- Integer Types (signed by default… unsigned with optional leading keyword)

C Type (Signed)	C Type (Unsigned)	Bytes	Bits	Signed Range	Unsigned Range
char	unsigned char	1	8	-128 to +127	0 to 255
short	unsigned short	2	16	-32768 to +32767	0 to 65535
int	unsigned int	4	32	-2 billion to +2 billion	0 to 4 billion
long	unsigned long	8	64	$-8*10^{18}$ to $+8*10^{18}$	0 to $16*10^{18}$

Text

- Text characters are usually represented with some kind of binary code (mapping of character to a binary number such as 'a' = 01100001 bin = 97 dec)
- ASCII = Traditionally an 8-bit code
  - How many combinations (i.e. characters)?
  - English only
- UNICODE = 16-bit code
  - How many combinations?
  - Most languages w/ an alphabet
- In C/C++ a single printing/text character must appear between single-quotes (')
  - Example: 'a', '!', 'Z'

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

http://www.theasciicode.com.ar/

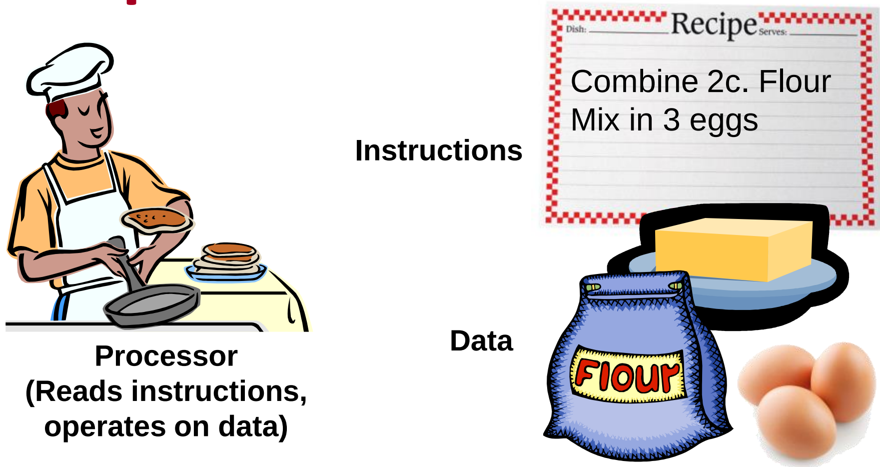
Text Strings

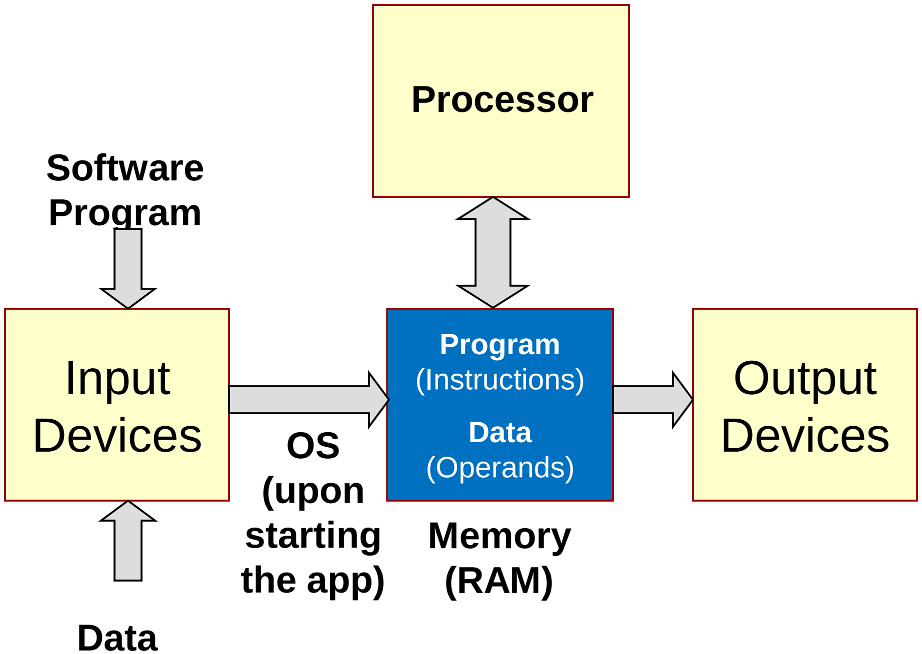
- To represent words and sentences we can use a **string** of characters
  - C++ uses double-quote (") to group the characters that are part of a string
- Example:
  - `"Hello\n"`
  - Each character is converted to ASCII equivalent
    - 'H' = 72, 'e' = 101, …
    - '\n' = Newline or Line Feed (LF) = Represents the non-printing character "Enter/Return" and moves the cursor to the start of the next line

DATA STORAGE & COMPUTER MEMORY

Computer Components

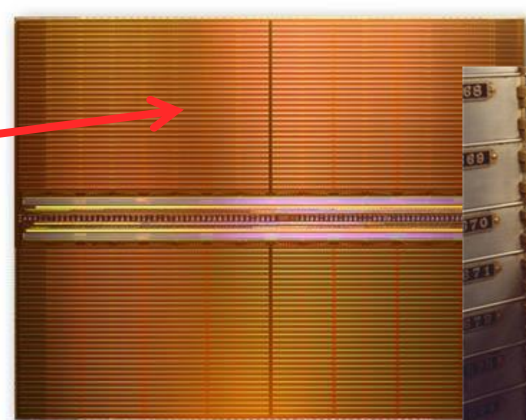
- Processor
  - Executes the program and performs all the operations
- Main Memory
  - Stores *data* and *program (instructions)*
  - RAM = read and write but volatile (lose values when power off)
- Let's look more at **memory**

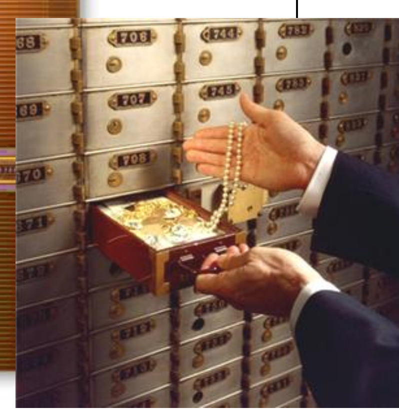




Memory

- Set of cells that each store a group of bits (usually, 1 byte = 8 bits)
- Unique address assigned to each cell
  – Used to reference the value in that location
- Analogy: Safe-deposit or mail boxes
  – Each has an identifying number and a value stored inside
  – The value can be an instruction, a number, a character, etc. (You the programmer must know what to expect and how to interpret it!)



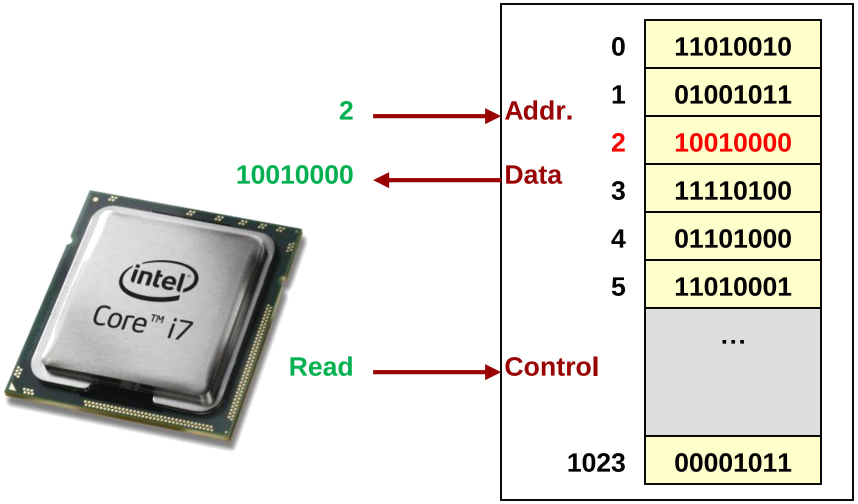


Address	Data
0	11010010
1	01001011
2	10010000
3	11110100
4	01101000
5	11010001
	...
1023	00001011

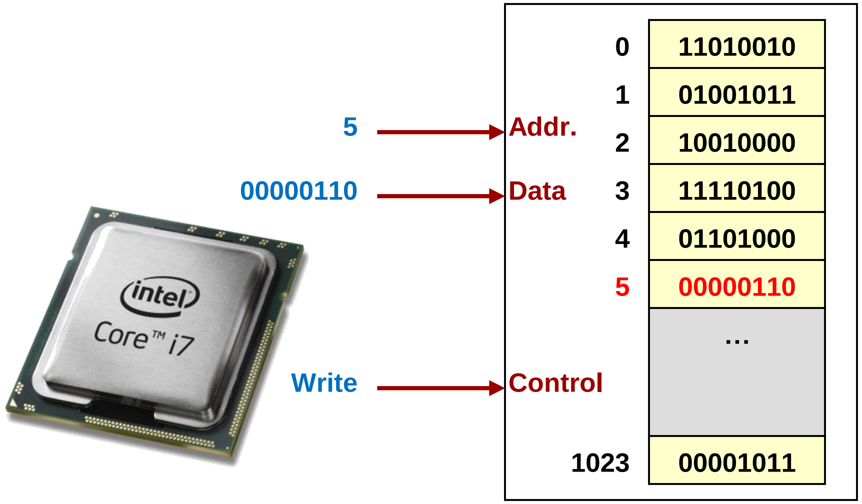
Memory Device

Memory Operations

- Memories perform 2 operations
  - Read: retrieves data value in a particular location (specified using the address)
    - You, the programmer, must know what type (integer, character, etc.) that data is.
  - Write: changes data in a location to a new value
- To perform these operations a set of address, data, and control inputs/outputs are used
  - Note: A group of wires/signals is referred to as a ‘bus’
  - Thus, we say that memories have an address, data, and control bus.



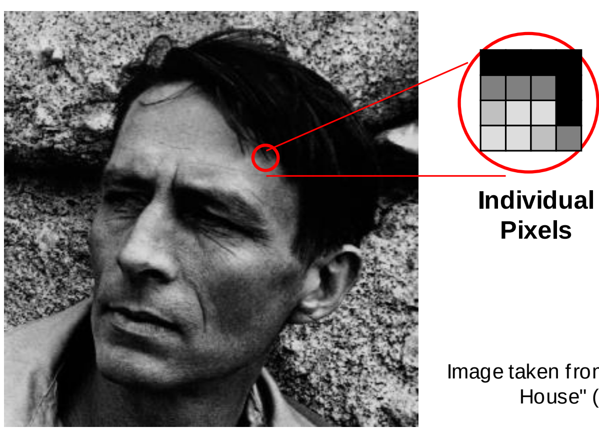
A Read Operation



A Write Operation

One At a Time

- Recall that while we see the image of a man, the computer "sees" a collection of numbers (aka pixels)?
- Now we can understand why
  - Every number is stored as bits in memory
  - Memory can only be accessed **one data value at a time**
- This limitation of accessing one value at a time leads to a fundamental issue of programming: How do we break abstract tasks into a sequence of "1 at a time" operations?



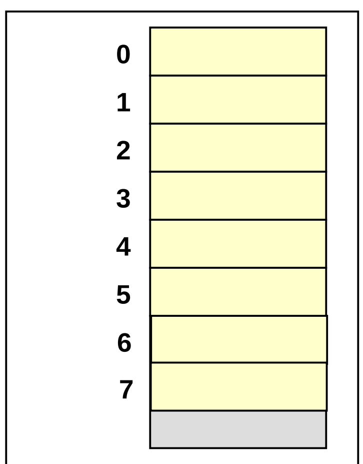
0	0	0	0
64	64	64	0
128	192	192	0
192	192	128	64

Image taken from the photo "Robin Jeffers at Ton House" (1927) by Edward Weston

Addr.	Data
0	0000 0000
1	0000 0000
2	0000 0000
3	0000 0000
4	0100 0000
5	0100 0000
6	0100 0000
7	0000 0000
8	1000 0000
	…

Exercise

- Show how "cs 102" would be stored in the memory below
  - Use decimal to represent each byte
- How do we indicate the string is done ("terminated")
  - With special NULL character (i.e. 0)



ASCII control characters			ASCII printable characters					
00	NULL	(Null character)	32	space	64	@	96	`
01	SOH	(Start of Header)	33	!	65	A	97	a
02	STX	(Start of Text)	34	"	66	B	98	b
03	ETX	(End of Text)	35	#	67	C	99	c
04	EOT	(End of Trans.)	36	$	68	D	100	d
05	ENQ	(Enquiry)	37	%	69	E	101	e
06	ACK	(Acknowledgement)	38	&	70	F	102	f
07	BEL	(Bell)	39	'	71	G	103	g
08	BS	(Backspace)	40	(	72	H	104	h
09	HT	(Horizontal Tab)	41	)	73	I	105	i
10	LF	(Line feed)	42	*	74	J	106	j
11	VT	(Vertical Tab)	43	+	75	K	107	k
12	FF	(Form feed)	44	,	76	L	108	l
13	CR	(Carriage return)	45	-	77	M	109	m
14	SO	(Shift Out)	46	.	78	N	110	n
15	SI	(Shift In)	47	/	79	O	111	o
16	DLE	(Data link escape)	48	0	80	P	112	p
17	DC1	(Device control 1)	49	1	81	Q	113	q
18	DC2	(Device control 2)	50	2	82	R	114	r
19	DC3	(Device control 3)	51	3	83	S	115	s
20	DC4	(Device control 4)	52	4	84	T	116	t
21	NAK	(Negative acknowl.)	53	5	85	U	117	u
22	SYN	(Synchronous idle)	54	6	86	V	118	v
23	ETB	(End of trans. block)	55	7	87	W	119	w
24	CAN	(Cancel)	56	8	88	X	120	x
25	EM	(End of medium)	57	9	89	Y	121	y
26	SUB	(Substitute)	58	:	90	Z	122	z
27	ESC	(Escape)	59	;	91	[	123	{
28	FS	(File separator)	60	<	92	\	124	|
29	GS	(Group separator)	61	=	93	]	125	}
30	RS	(Record separator)	62	>	94	^	126	~
31	US	(Unit separator)	63	?	95	_		
127	DEL	(Delete)						

Additional Resources

- https://www.youtube.com/watch?v=wgbV6DLVezo&feature=youtu.be