CS102 Unit 0b – Digital Representation and C++ Data Types and Constants

Mark Redekopp

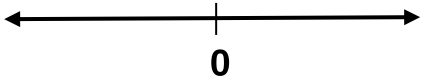
RANGE OF NUMBERS AND DATA TYPES

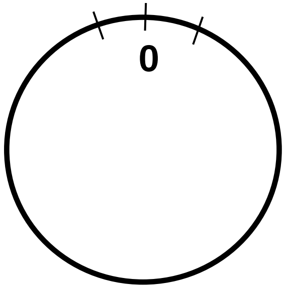
Finite Range of Numbers

- **Recall**: **EVERYTHING** in a computer is a **number**!
- Scenario: A hotel has 3-digit room numbers.
  - How many rooms can the hotel have?
  - What if the hotel uses 4-digit room numbers?
  - Range for n-digit room numbers?
- **Key Idea**: A fixed number of digits (or bits, for a computer), limits the range of numbers we can represent.
- What is 999+1?
  - 1000, obviously! Right!?
  - Well, if we limit ourselves to 3-digit numbers, then the answer is 000! We call this **overflow** and it is a common issue programmer's must account for.
- So, the number of digits available, determines the range of numbers that can be represented

3-digit Room Number

4-digit Room Number





Bits, Bytes, Words

- Computers store data as bits (binary digits) in units of memory with a fixed number of bits
- A single **bit** can only represent 1 and 0
- To represent more than just 2 values we need to use a combination / sequence of many bits
- Computer **hardware** (memory) defines common, easily accessible units of a fixed size:
  - A **byte** is defined as a group 8-bits
  - A **word** varies in size but is usually 32-bits (4 bytes)
- For n-bit numbers, the range of values we can represent is 0 to 2^n-1
  - For 8-bits, the range is 0 to 255.
  - For 32-bits, the range is 0 to 4,294,967,295

1

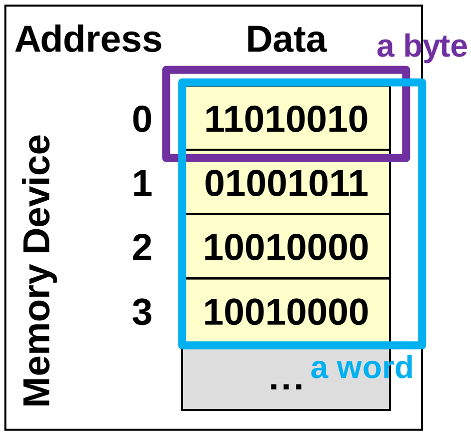
A bit

01000001

A byte (C++ char)

00101110 11010001
10110101 01110111

A "word" (C++ int)



Computer memory (storage) is broken into bytes (with 1 or more representing data values)

Finite Range of Binary

- 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:

0	= 0
1	= 1
1	

2^1 = 2 values

If n=2 bits:

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

2^2 = 4 values

If n=3 bits:

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
4	2	1	

2^3 = 8 values

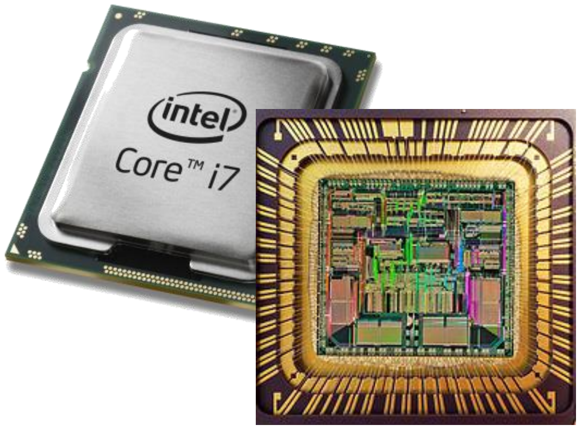
Given n bits, 2^n numbers can be made

Basic Computer Organization: Processor and Memory

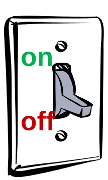
INTERLUDE

Digging Deeper

- So **why** do we have "bits" (that can only be 2 values) and **how** do we process and store them?
- Modern computer chips are made from ***billions*** 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**
  - The bit coming out of one transistor can control one or more other transistors creating complex processing chains that can perform functions like arithmetic



Transistor Acts as a Switch



High Voltage → **1**
+5V
+12V

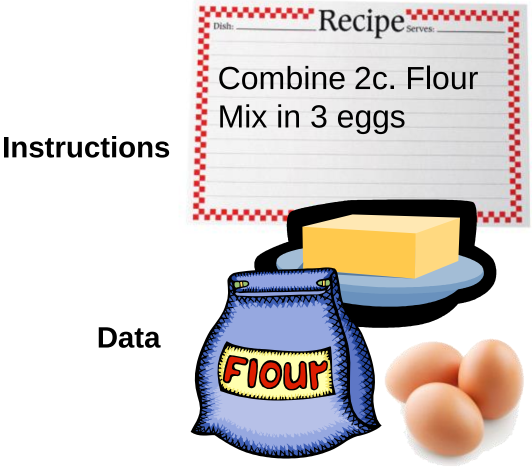
or

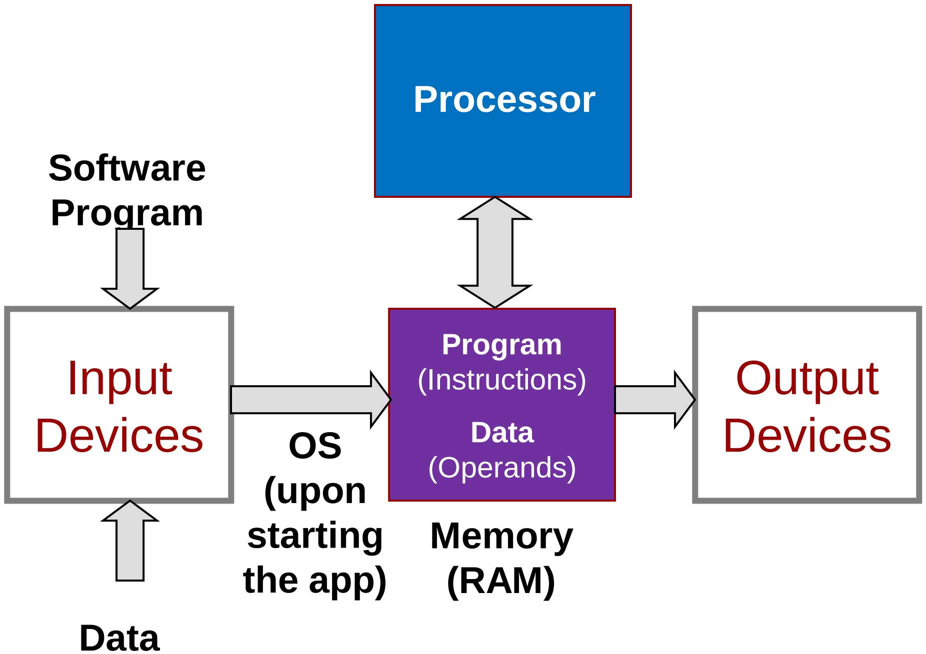
Low Voltage → **0**
0V
-12V

Computer Components

- Computer hardware circuits can be categorized into processor, memory, and I/O circuits
- If data is just bits that the processor manipulates with transistors, **where** do we store them when they are not being used?
- **Processor**
  - Executes the program and performs all the operations
- **Main Memory (aka RAM)**
  - Stores *data* and *program (instructions)*
  - Loses data when power is disconnected
- Let's look at **memory**

Processor (Reads instructions, operates on data)

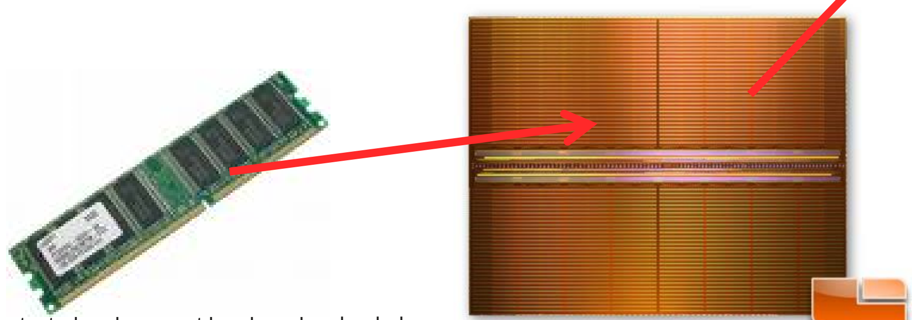




Memory

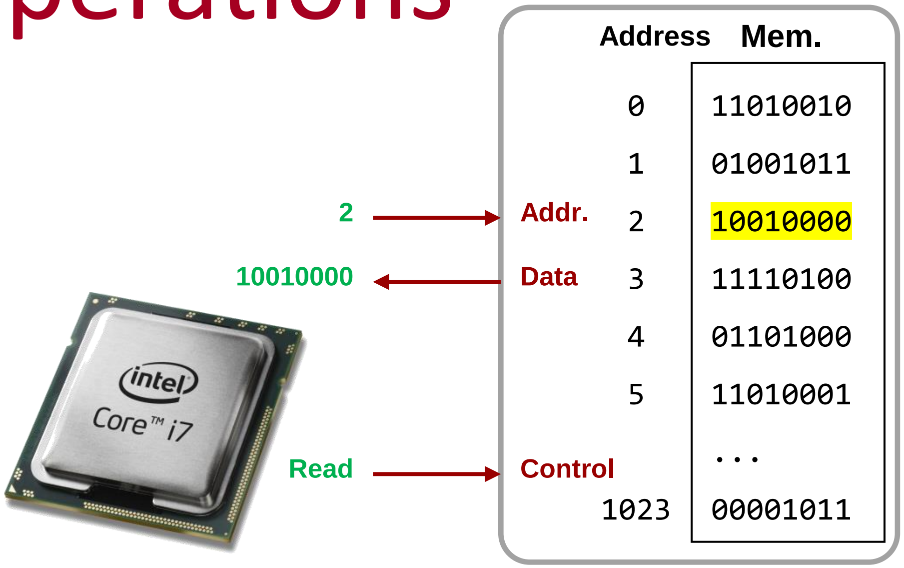
- Also uses transistors, but in a different way that allows the transistors to help "remember" bits
- Broken into "cells" that each store a group of bits (usually, 1 byte = 8 bits) and is accessed via a unique number (aka "address")
- The address is used to reference the value a given location
- Analogy: Safe-deposit or mailboxes
  - 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…no meta-information is present to tell you how to interpret the bits)

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

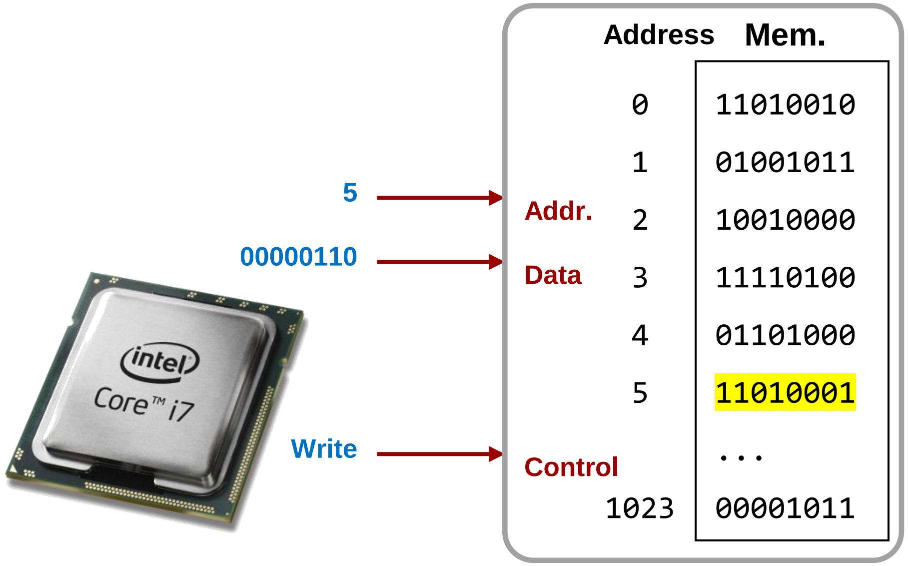


Memory Operations

- Memories perform 2 operations
  - Read: retrieves data value in a particular location (specified using the address)
  - 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.



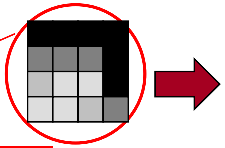
A Read Operation



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?



Individual Pixels

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

Address	Mem.
7420	00
7421	00
7422	00
7423	00
7424	64
7425	64
…	…
7434	128
7435	64

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

C++ DATA TYPES AND CONSTANTS

Motivation for Data Types

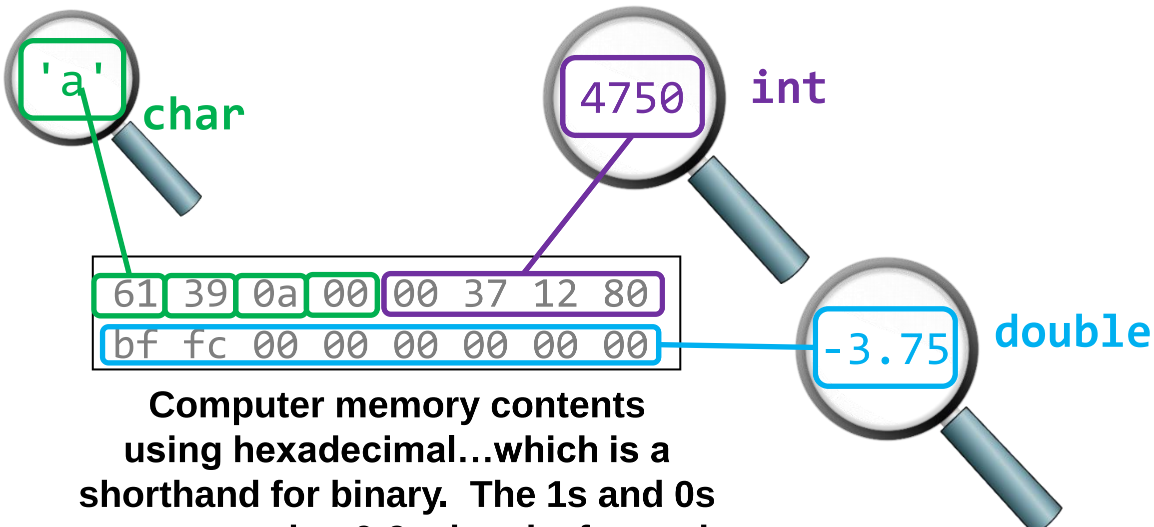
- So information is stored in group of bits (bytes and words)
- How many data values are stored in the memory below (where does one value stop and another start) and what are their values?

```
61 39 0a 00 00 37 12 80
bf fc 00 00 00 00 00 00
```

Computer memory contents using hexadecimal…which is a shorthand for binary. The 1s and 0s are converted to 0-9,a,b,c,d,e,f to make it easier for humans to read

Motivation for Data Types

- So information is stored in group of bits (bytes and words)
- How many data values are stored in the memory below (where does one value stop and another start) and what are their values?
- C/C++ **types** indicate how many bits (bytes) of **storage** (memory) are required and how to **interpret** the number being stored



Computer memory contents using hexadecimal…which is a shorthand for binary. The 1s and 0s are converted to 0-9,a,b,c,d,e,f to make it easier for humans to read

C/C++ Data Types

- C/C++ **types** indicate how many bits (bytes) of **storage** (memory) are required and how to **interpret** the number being stored
- **Integer types**
  - **int, unsigned int** (and, technically, char - more explanation later)
- **Floating point types -** Very large 6.02E23 & very small numbers 6.626E-34 (i.e. an attempt to represent rational/real numbers)
  - float or double (in general, prefer **double** over float as it has a greater range of expressivity)
- **String/Text types**
  - char, char arrays, strings
- **Boolean type**
  - bool (true / false)

- Let's look at how to write constants (aka "literals") and declare variables of these types.

Constants (aka Literals)

- Integer: `496, 10005, -234`
- Double: `12.0, -16., 0.23, 6.02E23, 4e-2`
  - Both very large and very small numbers (i.e. fractions/decimals)
- Characters (char type): enclosed in **single quotes (')**
  - Printing characters: `'a', '5', 'B', '!'`
  - Each quoted value is converted to appropriate ASCII number (e.g. 'a' => 97)
  - Non-printing special characters use "escape" sequences (i.e. preceded by a \ ): `'\n'` (newline/enter), `'\t'` (tab) , `'\\'` (slash), `'\''` (apostrophe)
- C-Strings (Note: there is also a C++ string type...)
  - **0 or more** characters between **double quotes (")**

    `"hi1\n", "12345", "b", "\tAns. is %d"`
  - Ends with a `'\0'`=0 (aka NULL character) added as the last byte/character to allow code to delimit the end of the string
- Boolean (C++ only): `false, true`
  - Physical representation: 0 = false, Non-zero (1, -5, 300) = true

C/C++ handling of single characters and strings is different than most other languages and a major source of confusion in C++.

Address	Mem.	
7420	104	'h'
7421	105	'i'
7422	49	'1'
7423	10	'\n' newline
7424	00	'\0' null
7425	35	'#'
7426	100	'd'
...	…	

C-String Example (Memory Layout)

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` or `'\0'`)

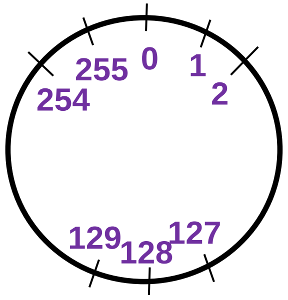
Address	Mem.
7420	
7421	
7422	
7423	
7424	
7425	
7426	
7427	
...	

ASCII control characters		
00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

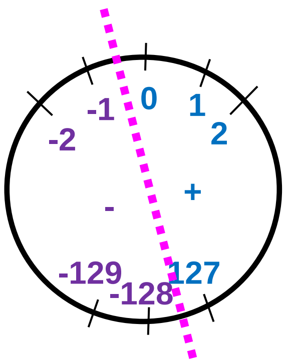
ASCII printable characters					
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	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	_		

Signed and Unsigned Integer Types

- If we have a finite range of numbers (**2^n**) that we can make with **n** bits, what values should they correspond to?
- C++ defines both "unsigned" and "signed" integer types
- "**unsigned**" integer types use all bit combinations for POSITIVE (natural) numbers (0 to 2^n-1)
- "**signed**" integer types split the combinations with half being positive numbers and half being negative
- C++ also defines other intermediate sizes (1-, 2-, 4-, 8-byte integer types) that have more range but use more memory



Unsigned (all positive) Types
`unsigned int`
`unsigned long`



Signed (pos. or neg.) Types
`int`
`long`

C/C++ Integer Data Types

- Integer variable types
  - An unsigned (positive-only...including 0) number
  - A signed (positive or negative) number

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 long	unsigned long long (aka size_t)	8	64	$-8*10^{18}$ to $+8*10^{18}$	0 to $16*10^{18}$

***These are the three integer types we will use 99% of the time**

C/C++ Floating Point Types

- `float` and `double` types:
  - Allow decimal representation (e.g. 6.125) as well as very large integers (+6.023E23)

C Type	Bytes	Bits	Range
float	4	32	±7 significant digits * $10^{+/-38}$
double	**8**	**64**	**±16 significant digits * $10^{+/-308}$**

- Prefer `double` over `float`
  - Many compilers will upgrade floats to doubles anyhow
- Don't use floating-point if you don't need to
  - It suffers from rounding error
  - Some additional time overhead to perform arithmetic operations

Additional Resources

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