CMPT 130

Introduction to Computing Science and Programming I

What this course is about...

- Introduces basic ideas of computing science
- Describes the working principles of a computing machine (computer)
  - Information representation, Storage, and Processing
- Introduces basic concepts of communicating with a computer
  - Programming (**C++ Programming Language**)
- Introduces computational formulation of methods to solve given problems: Pseudocodes and Algorithms

How Does a Computer work?

- A computer is a **DIGITAL ELECTRONIC** machine!
  - As such, it understands only ‘certain’ **NUMBERS**!!!
- Really?
  - How are we then able to work with alphabets, symbols, images, videos, sound and much more things on our computers?
- Answer!
  - Everything in a computer is represented by unique combinations of numbers. (Example the letter ‘**a**’ is nothing but 97. We will come to this later)

Decimal Number System

- Let us begin with the Decimal Number System.
  - Numbers we (humans) use on our daily life
  - Ten unique digits (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9)
  - Every other number is represented by combining these unique digits
  - Why exactly 10 digits? Because we have 10 fingers and before the days of calculators we used our fingers for counting
  - But how many fingers does a computer have?

Decimal Number System (continued)

- How do we represent numbers outside the range of the basic digits in the decimal number system?
  - Write a number as a string of symbols.
  - Each symbol is one of the ten digits (0 – 9)
  - Interpret the number by multiplying each digit by a power of 10 associated with that digit's position
- Example:
  - The number $2578 = 2 \cdot 10^3 + 5 \cdot 10^2 + 7 \cdot 10^1 + 8 \cdot 10^0$

Decimal Number System (cont.)

- Decimal number system is called base-10 system
- The right most digit is least significant digit (exponent 0) and the left most digit is the most significant digit
- The exponent associated with each digit increments by 1 as we go from right most digit to the left
- Leading zeros do not have effect: 00762 = 762
- Negative numbers are represented by prefixing the negative sign (-) to a number

What about a computer?

- Now, consider computers: made up of transistors.
- Transistors can only be **ON** or **OFF**.
- Therefore a computer can understand only two states (ON or OFF states of its millions of transistors)
- These two states are represented by **0** and **1**
- Thus a computer can understand only two numbers 0 and 1. (Computer has only two fingers!)
- Therefore we use binary number system (base-2) to represent any information in computers

Unsigned Binary Number System

- Consider the binary number **1101**. What number in decimal system does it represent?
  - It represents: $1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8+4+0+1 = 13$
- In the binary number system
  - Each symbol is called a BIT
  - In decimal the base is 10, in binary it is 2
  - As in decimal, interpreting a binary string entails multiplying each bit by a power of 2 associated with that bit's position

Examples

- Express the following binary numbers in decimal: **1111**, **10001**, **11100101**
- Solution

  1111 = $1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8+4+2+1 = 15$

  10001 = $1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 16+0+0+0+1 = 17$

  11100101 = $1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 128 + 64 + 32 + 4 + 1 = 229$

- **Remember**: powers of 2 start at 0 for the right most bit and increment by 1 as we go left

Expressing Unsigned Decimal Numbers in Binary

- So, we saw that **11100101** = 229. But how do we go from 229 to express it as binary?

Procedure to convert decimal to binary

Step 1: Divide the decimal by 2 and record the remainder.

Step 2: Continue dividing the newest quotient obtained by 2; recording the remainders until a quotient of 0 is obtained.

Step 3: The binary representation is then the remainders recorded; starting from the last remainder going all the way to the first remainder in the order they were recorded.

Expressing Unsigned Decimal Numbers in Binary (cont.)

- Example: convert 229 to binary
- Solution:

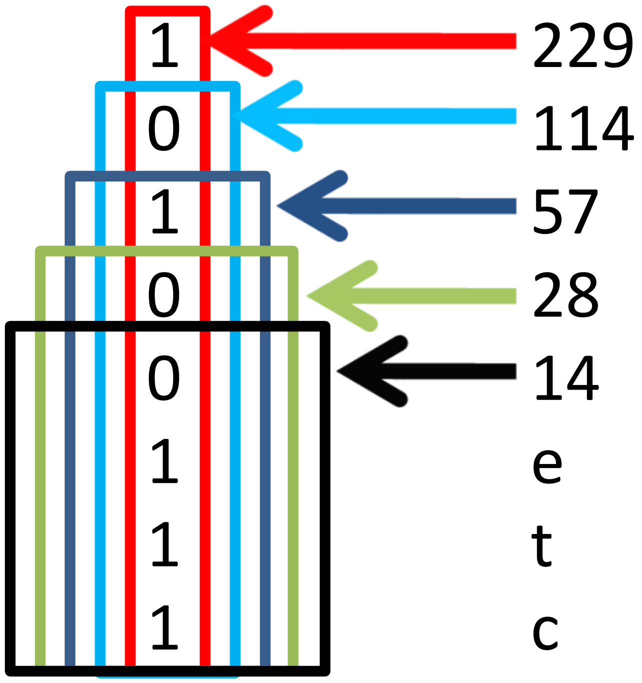
Operation	**Quotient**	**Remainder**
229 / 2	114	1 ... Least significant bit
114 / 2	57	0
57 / 2	28	1
28 / 2	14	0
14 / 2	7	0
7 / 2	3	1
3 / 2	1	1
1 / 2	0	1 ... Most significant bit

- The answer is therefore **11100101** as expected
- **Question**: What are the binary representations of 57, 7, 114?

Expressing Unsigned Decimal Numbers in Binary (cont.)

- Observation

Operation	**Quotients**	**Remainders**
229 / 2	114	1
114 / 2	57	0
57 / 2	28	1
28 / 2	14	0
14 / 2	7	0
7 / 2	3	1
3 / 2	1	1
1 / 2	0	1



Bit Patterns

- Grouping bits together yields bit patterns common in computers.

– Group of 1 bit	Bit
– Group of 4 bits	Nibble
– Group of 8 bits	Byte
– Group of 16 bits	Word
– Group of 32 bits	Double Word

- Example: the decimal 13 is **1101** in Nibble, **00001101** in Byte and **0000000000001101** in Word

Representing Unsigned Decimal Numbers in Bit Patterns

- Consider the Nibble (4-bit pattern). What are the decimal numbers that can be represented with it?
  - It is easy to see that the numbers 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 and 15 can be represented with a Nibble because their binary representations are: **0000**, **0001**, **0010**, **0011**, **0100**, **0101**, **0110**, **0111**, **1000**, **1001**, **1010**, **1011**, **1100**, **1101**, **1110**, and **1111** respectively. (16 symbols!)
  - The decimal 16 can not be represented by a Nibble because its binary representation **10000** needs five bits which the Nibble can not handle

Representing Unsigned Decimal Numbers in Bit Patterns (cont.)

- Generally, a bit pattern of **n** bits can represent the decimal numbers **0** through **2^n-1**
- Thus
  - A Bit can represent the decimals 0 to 2^1-1 = 0 to 1
  - A Nibble can represent the decimals 0 to 2^4-1 = 0 to 15
  - A Byte can represent the decimals 0 to 2^8-1 = 0 to 255
- **Remark**: The number of decimal numbers that can be represented by a bit, nibble and byte is respectively 2, 16 and 256. (Don’t forget to count the decimal 0!)
- Generally, **n** bit pattern can represent **2^n** decimal numbers. These decimals are **0, 1, 2,...., (2^n-1)**

Negative Decimal Numbers

- So far, we have seen only positive numbers and zero. Their binary representations are called **unsigned binary numbers**
- But what about negative numbers (signed numbers)?
  - In Decimal Number System, prefixing a number with the symbol minus (-) sign makes it negative
  - In a computer, there is no notion of minus sign. The only basic units are 0 and 1 bits
- One way of representing negative numbers is with **Sign and Magnitude Representation** where the left most bit is reserved for sign

Sign and Magnitude Representation

- Given a bit pattern (nibble, byte, word,...), reserve the left most bit for sign and the remaining bits for the actual decimal number (magnitude)

0 → positive and 1 → negative

- Therefore considering the nibble, we have

0000	0	1000	-0
0001	1	1001	-1
0010	2	1010	-2
0011	3	1011	-3
0100	4	1100	-4
0101	5	1101	-5
0110	6	1110	-6
0111	7	1111	-7

Example

- Give the sign and magnitude representation of the signed decimal number -45
  a) As a seven bit pattern
  b) As a byte
  c) As a word
- Solution: Since the unsigned binary representation of the magnitude, which is 45, is given by 101101; we have
  a) In seven bit, -45 as a sign and magnitude representation is 1101101
  b) As a byte, it will 10101101
  c) As a word, it will be 1000000000101101

Sign and Magnitude Representation

- How many decimal numbers can be represented by a given bit pattern using sign and magnitude representation?
- As shown above, a nibble can represent the signed integers -7 up to +7. Therefore it can represent 15 signed integers. Count them!
- Generally, an **n-bit** pattern can represent the signed integers **$-(2^{n-1}-1)$ up to $(2^{n-1}-1)$**

Sign and Magnitude Representation (cont.)

- Drawbacks
  - It represents the decimal zero as two different bit patters
  - In order to do arithmetic on such binary bit patterns, we first need to identify which number is negative and which one is positive for correct arithmetic. This is extra work for a computer
  - After doing an arithmetic, one would need to assert the sign bit is correctly set depending on the result. Again, extra work for the computer

Problem

- How can we represent signed integers in binary such that
  - The system represents zero as a unique bit pattern
  - Arithmetic on the system follows a standard arithmetic procedure with no additional task to handle signs
  - The result of any arithmetic will have the correct sign
- Solution
  - We use **Two's Complement Representation**!

Two’s Complement Representation

- An elegant way to avoid the problems of signs and ambiguity in representing zero.
- All modern computers represent numbers with two’s complement notation
- To find the two’s complement of a given binary number, first flip all the digits and then add 1 to the result
- But how do we perform addition in binary number system?

Two’s Complement Representation (cont.)

- Binary addition (single digit)

$\begin{array}{r} 0 \\ +\,0 \\ \hline 0 \end{array}$	$\begin{array}{r} 1 \\ +\,0 \\ \hline 1 \end{array}$	$\begin{array}{r} 0 \\ +\,1 \\ \hline 1 \end{array}$	$\begin{array}{r} 1 \\ +\,1 \\ \hline 10 \end{array}$

- Binary addition (bit patterns)

$\begin{array}{r} 1101 \\ +\,1011 \\ \hline 11000 \end{array}$	$\begin{array}{r} 111101 \\ +\,101 \\ \hline 1000010 \end{array}$

- Note: 1 + 1 = 0 with a carry 1

Two's Complement Representation (cont.)

- Now, we can easily compute the two's complement of binary numbers
  - **Example**: What is the two's complement of **1011**?
  - **Solution**
    - Step 1. Flip all bits to get **0100**
    - Step 2. Add 1 to the result to get **0100** + **1** = **0101**.
  - Therefore, the two's complement of **1011** is **0101**.
- **Remark**: The two's complement of a binary number must be expressed in the same bit pattern in order to get a correct interpretation of the result. See below.

Two's Complement Representation (cont.)

- **Example**: Given the nibble **0000**, find its two complement.
  - **Solution**
    - Step 1. Flip all bits to get **1111**
    - Step 2. Add 1 to the result to get **1111** + **1** = **10000**.
  - **Remark**: The left most carry bit is discarded because it goes out of the capacity of a nibble. Therefore, the two's complement of **0000** is **0000**.
- **Example**: Given the nibble **1111**, find its two complement.
  - **Solution**
    - Step 1. Flip all bits to get **0000**
    - Step 2. Add 1 to the result to get **0000** + **1** = **0001**.
  - Therefore, the two's complement of **1111** is **0001**.

Representation of Integers in Two’s Complement

- How do we represent signed integers in two’s complement?
- Solution
  - **Positive Decimal Numbers and Zero**

    Step 1. Their two’s complement representation is the same as their unsigned binary representation

  - **Negative Decimal Numbers**

    Step 1. Find the unsigned binary representation of the decimal number without the negative sign

    Step 2. Flip all the bits in Step 1

    Step 3. Add 1 to the result in Step 2

Examples

- Find the binary representation of 53 in two’s complement representation as a byte
  - Solution
    - The unsigned binary representation of 53 as a byte is **00110101**. This is also its two’s complement representation.
- Find the binary representation of -53 in two’s complement as a byte
  - Solution
    - **Step 1**. Find binary representation of 53. It is **00110101**
    - **Step2**. Flip all bits to get **11001010**
    - **Step 3**. Add 1 to the result to get **11001011**
- What about -103 in two's complement as a byte pattern?
  - Solution: Steps 1 through 3 will respectively give the results: **01100111**, **10011000**, and **10011001**. Therefore the answer is **10011001**

Examples (cont.)

- Add 53 with -53 in two's complement using a byte.
  - Solution:

```
      53   is        00110101
+    -53   is        11001011
---------------------------------
       0            100000000
```

- Once again, the left most carry bit will be **discarded** as it will go **out of the capacity of a byte**. Therefore the answer will be **zero** as expected!

Closer Look at Two's Complement Representation

- Now, let us see which **signed** integers can be represented by a given bit pattern

- For this purpose, let us consider a nibble

Decimal	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Binary	**0000**	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**
Decimal	**-1**	**-2**	**-3**	**-4**	**-5**	**-6**	**-7**	**-8**
Binary	**1111**	**1110**	**1101**	**1100**	**1011**	**1010**	**1001**	**1000**

- Notice that the bit pattern **1000** could have been used for **+8** or **-8** equally right. However, computer scientists decided to assign it **-8** for reasons that will be evident soon.

Properties of Two's Complement Representation

1. The two's complement of **positive** decimal numbers and **zero** always start with a **0** bit. While that of **negative** decimal numbers start with a **1** bit
2. A nibble can represent the signed integers -8 to +7 in two's complement, as shown above. Similarly, a byte can represent signed integers -128 to +127 in two's complement
3. Generally, an **n**-bit pattern can represent the signed integers $\mathbf{-(2^{n-1})}$ to $\mathbf{+(2^{n-1}-1)}$

Conversion from Two’s Complement Binary Representation to Decimal Numbers

- In order to convert a given binary number in two’s complement to decimal, follow these steps:
  - If it starts with a **0**, then it is positive or zero integer.
    1. Convert it to decimal by expansion with powers of 2
    2. The resulting decimal is the required answer
  - Else if it starts with a **1**, then it is negative integer.
    1. Find the two’s complement of the given binary number
    2. Convert it to decimal by expansion with powers of 2
    3. The negative of the resulting decimal is the required answer

Examples

- Find the integers represented by the following binary numbers represented in two's complement: **010101**, **11101101**, and **10000000**.
  - **010101** is positive. Expanding it results 21. Therefore the answer is 21.
  - **11101101** is negative. Its two complement is **00010011**. This binary corresponds to the decimal 19. Therefore the answer is -19.
  - **10000000** is negative. Its two complement is **10000000**. This binary corresponds to the decimal 128. Therefore the answer is -128.

Examples (cont.)

- Consider the decimal number 5 in Byte pattern represented in two’s complement
  - Its binary representation is **00000101**
  - Its two’s complement is **11111011** which is -5
- Now consider the decimal number -5 in byte pattern represented in two’s complement
  - Its binary representation is **11111011**
  - Its two’s complement is **00000101** which is 5
- Thus, the two’s complement of two’s complement of a binary number is the binary number itself!

Arithmetic of Integers represented in Two’s Complement Representation

- Let us perform the operations 3+-5, -3+5 and -3+5 in two’s complement in Nibble and Byte forms:

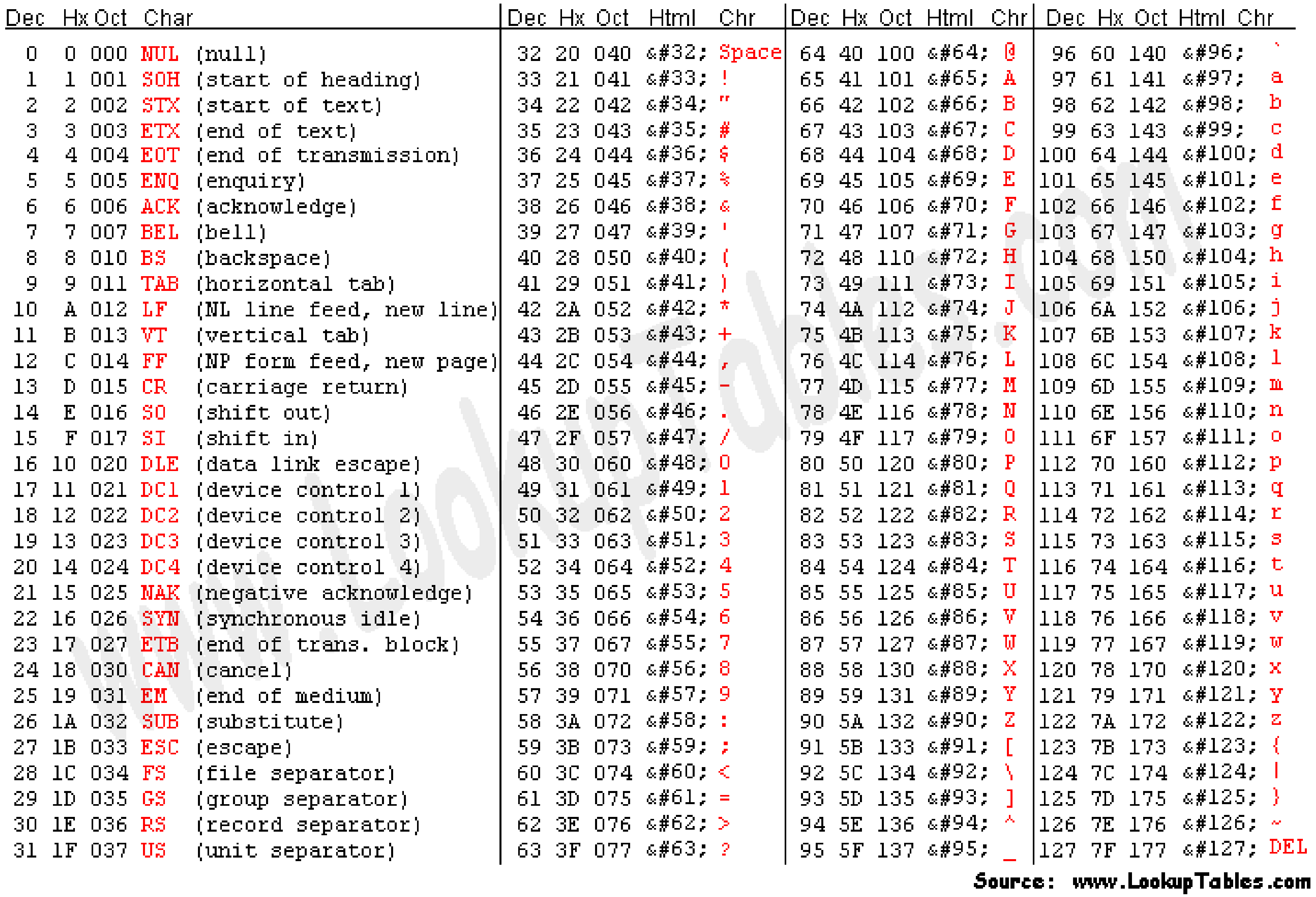
3 0011	**-3 1101**	**-3 11111101**
+ -5 1011	**+ 5 0101**	**+ 5 00000101**
-2 1110 ➔ -2	**2 10010 ➔ 2**	**2 100000010 ➔ 2**

- The discarded carry bits are shown in red.
- When a carry over goes beyond the capacity of the bit pattern under consideration, then the operation is said to give rise to an **overflow**

Information Representation in Computers

- So far, we have seen how signed or unsigned numbers are represented in computers using unsigned binary representation, sign and magnitude representation, and two’s complement representation
- But what about alphabets (a,b,c,..; A, B, C,...), digits (0, 1, 2,...), symbols (., :, ;, {,], (, &, α, β, ...) etc
- Solution
  - These objects are encoded with an agreed code as unsigned binary numbers. **Ascii code** is used mostly. See http://www.asciitable.com/

Ascii Code Table



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Information Representation in Computers (cont.)

- **Ascii Code**
  - Stands for: **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
  - It uses 8-bit (byte) to represent most symbols used in English language (alphabets, punctuation marks, symbols, digits,...)
  - **It uses unsigned binary representation in binary**
- In order to extend Ascii code for other than English language, **Unicode** (16-bit) is used
- Recently, **ISO code** (32-bit) is also used to represent even more symbols: everything in the world

Information Representation in Computers (cont.)

- Consider the message “Hello 123”. How do we represent it in the computer?

Symbol	Ascii Code (Decimal)	Ascii Code (8-bit Binary)
H	72	**01001000**
e	101	**01100101**
l	108	**01101100**
l	108	**01101100**
o	111	**01101111**
	32	**00100000**
1	49	**00110001**
2	50	**00110010**
3	51	**00110011**

- Therefore the bit sequence **010010000110010101101100011011000110111100100000001100010011001000110011** represents the message “Hello 123”

Putting It All Together

- All information stored and processed in a computer is represented with binary digits (bits)
- Sequence of binary bits can represent signed or unsigned integers, floating numbers, symbols, instructions,... BUT their interpretation schemes are different. Example
  - The binary bits sequence 11001010 may represent
    - The integer **202** (unsigned binary representation), or
    - The integer **-74** (sign and magnitude representation), or
    - The integer **-54** (two's complement representation), or
    - The character ╩ (Ascii code)

Putting It All Together(cont.)

- How about floating numbers (numbers with decimal point; example 1.5, -56.3, 2.07, 25.0). How are they represented in a computer?
  - Such types of numbers are represented in a completely different way. Not discussed in this course!
- It is the responsibility of a programming language to keep track of what information a given sequence of bits represents (signed numbers, unsigned numbers, floating numbers, Ascii code for symbols, instructions,...)

Hexadecimal Number Notation

- While the binary number system is the fundamental working system in computers, it is very difficult to read and write for us humans
- We can however group 4-bits (nibbles) together and write them as hexadecimal digits for easier reading and writing
- Hexadecimal number is a number system of base 16
- The basic hexadecimal digits are

  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F

Hexadecimal Number Notation (cont.)

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

- Therefore

  $(10001011)_{\mathbf{binary}}$ becomes $(8B)_{\mathbf{hex}}$

  $(11101111)_{\mathbf{binary}}$ becomes $(EF)_{\mathbf{hex}}$

  $(A75C)_{\mathbf{hex}}$ becomes $(1010011101011100)_{\mathbf{binary}}$

The Memory Unit of a Computer

- Let us start with a given File sitting on the desktop of your computer. Assume the size of this File is 25KB (Kilo-Bytes)? What does it mean?
  - First of kilo is a metric prefix that stands for 2^{10}. Therefore the size of a given File is 25KB is equivalent to saying $25*2^{10}$ = 25,600Bytes. Since a Byte is 8 bits, this means that the File contains 25,600*8 = 204,800 bits (0s and 1s) inside it.
- Similarly Mega stands for 2^{20} and Giga stands for 2^{30}.
- Next, consider the random access memory (RAM) of a computer. It is customary to say a given laptop has 512MB of RAM. This means its RAM is $512*2^{20}$ Bytes long. This means the RAM can store a maximum of $512*8*2^{20}$ bits at once.