01001110010000110101001101010101

Number Systems

4e435355

1,313,035,093

ECGR2181

Lecture Notes 2

1.525×2^{29}

Reading: Chapter 2

How do we represent data in a computer?

At the lowest level, a computer is an electronic machine.

- works by controlling the flow of electrons

Easy to recognize two conditions:

1. presence of a voltage – we'll call this state "1"
2. absence of a voltage – we'll call this state "0"

Could base state on *value* of voltage,
but control and detection circuits more complex.

- compare turning on a light switch to measuring or regulating voltage

We'll see examples of these circuits in the next chapter.

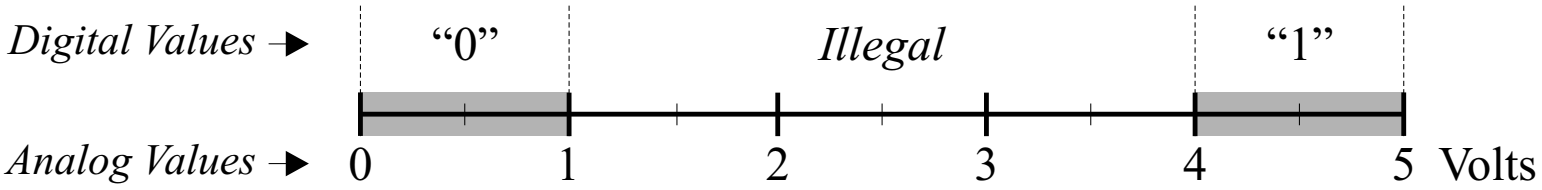
Computer is a binary digital system.

Digital system:

- finite number of symbols

Binary (base two) system:

- has two states: 0 and 1



Basic unit of information is the *binary digit*, or ***bit***.

Values with more than two states require multiple bits.

- A collection of two bits has four possible states:
  00, 01, 10, 11
- A collection of three bits has eight possible states:
- *A collection of **n** bits has **2^n** possible states.*

What kinds of data do we need to represent?

- Numbers – signed, unsigned, integers, floating point, complex, rational, irrational, …
- Text – characters, strings, …
- Images – pixels, colors, shapes, …
- Sound
- Logical – true, false
- Instructions
- …

Data type:

- *representation* and *operations* within the computer

We'll start with numbers…

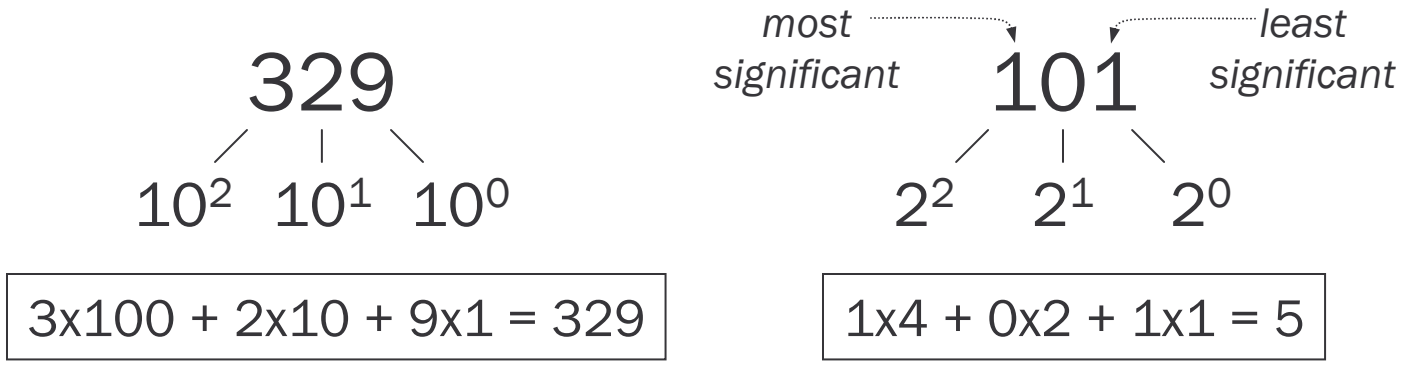
Unsigned Integers

Non-positional notation

- could represent a number (“5”) with a string of ones (“11111”)
- problems?

Weighted positional notation

- like decimal numbers: “329”
- “3” is worth 300, because of its position, while “9” is only worth 9



Unsigned Integers (cont.)

An ***n***-bit unsigned integer represents 2^n values: from 0 to 2^n-1.

2^2	2^1	2^0	Decimal
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

Unsigned Integer Practice

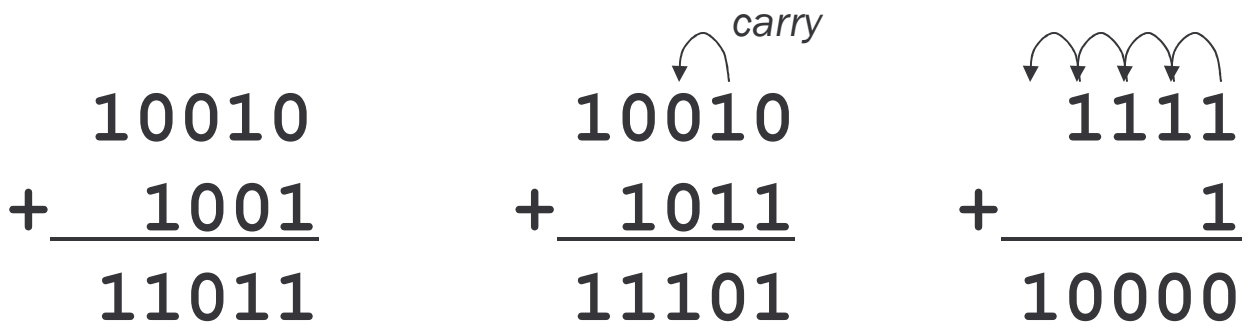
What is the decimal value of the unsigned binary number "1101"?

What is the binary representation of decimal 10?

Unsigned Binary Arithmetic

Base-2 addition – just like base-10!

– add from right to left, propagating carry



Unsigned Binary Arithmetic Practice

Base-2 addition – just like base-10!

- add from right to left, propagating carry

```
  10111          10111
+  1111        -  1111
```

Subtraction, multiplication, division,...

Signed Integers

With n bits, we have 2^n distinct values.

- assign about half to positive integers (1 through 2^{n-1}) and about half to negative (- 2^{n-1} through -1)
- that leaves two values: one for 0, and one extra

Positive integers

- just like unsigned – zero in most significant bit
  `00101 = 5`

Negative integers

- sign-magnitude – set top bit to show negative, bottom as in unsigned
  `10101 = -5`
- one's complement – flip every bit to represent negative
  `11010 = -5`
- in either case, most significant bit indicates sign: 0=positive, 1=negative

Is there a better way?

Problems with sign-magnitude and 1's complement

- two representations of zero (+0 and –0)
- arithmetic circuits are complex
  - How to add two sign-magnitude numbers?
    - e.g., try 2 + (-3)
  - How to add to one's complement numbers?
    - e.g., try 4 + (-3)

There is a great representation of negative numbers that uses the analogy of an automobile odometer.

Odometer numbers

Consider an odometer of a car at a location on a street:

0	0	1	3

Go 3 miles in reverse and it reads:

0	0	1	0

- **Same as "subtracting 3" or "adding -3"**
- **What happens when...**

0	0	0	0

Go 3 miles in reverse and it reads:

9	9	9	7

- **As far as the odometer is concerned, 9997 = -3**
- **<u>Note that fixed-width binary is very similar to odometer numbers in its limitations</u>**
- **Can the same representation be used?**
  - **0000_2-0001_2="1111_2" or -1**
  - **0000_2-0011_2="1101_2" or -3**
- **This is called "2's complement"**

Two's Complement

Two's complement representation developed to make circuits easy for arithmetic.

- for each positive number (X), assign value to its negative (-X), such that X + (-X) = 0 with "normal" addition, ignoring carry out

```
  00101  (5)
+ 11011  (-5)
  00000  (0)
```

```
  01001  (9)
+        (-9)
  00000  (0)
```

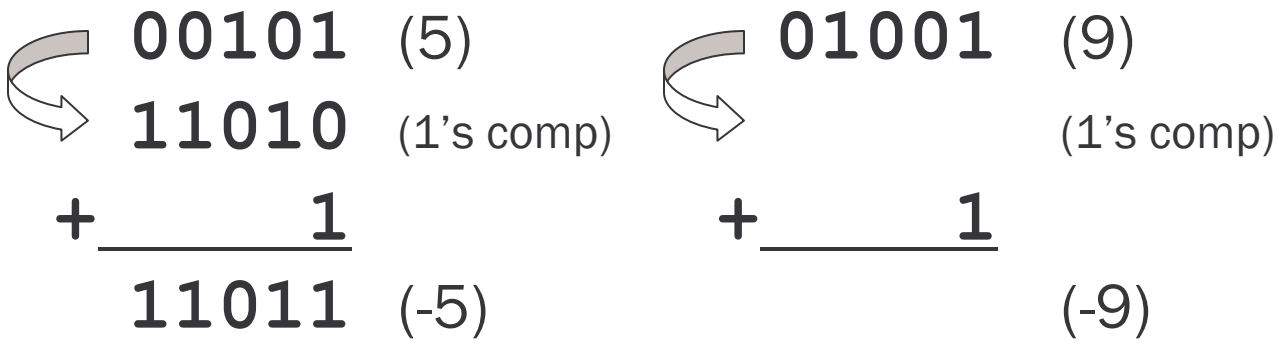
Two's Complement Representation

If number is positive or zero,

- normal binary representation, zeroes in upper bit(s)

If number is negative,

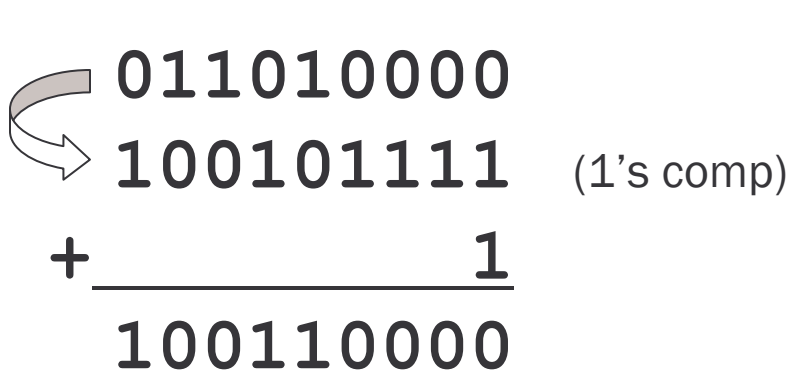
- start with positive number
- flip every bit (i.e., take the one's complement)
- then add one

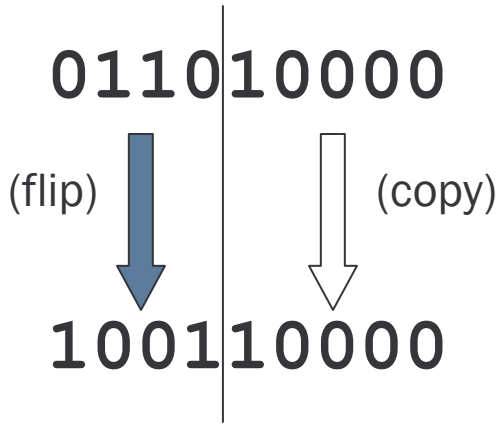


Two's Complement Shortcut

To take the two's complement of a number:

- copy bits from right to left until (and including) the first "1"
- flip remaining bits to the left





Two's Complement Signed Integers

MS bit is sign bit – it has weight -2^{n-1}.

Range of an n-bit number: -2^{n-1} through $2^{n-1} - 1$.

- The most negative number (-2^{n-1}) has no positive counterpart.

-2^3	2^2	2^1	2^0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

-2^3	2^2	2^1	2^0	
1	0	0	0	-8
1	0	0	1	-7
1	0	1	0	-6
1	0	1	1	-5
1	1	0	0	-4
1	1	0	1	-3
1	1	1	0	-2
1	1	1	1	-1

Two's Complement Practice

Show the two's complement representation of the decimal number -6.

Converting Binary (2's C) to Decimal

1. If leading bit is one, take two's complement to get a positive number.
2. Add powers of 2 that have "1" in the corresponding bit positions.
3. If original number was negative, add a minus sign.

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
(K) 10	1024
(M) 20	1048576

$$X = 01101000_{two}$$
$$= 2^6+2^5+2^3 = 64+32+8$$
$$= 104_{ten}$$

Assuming 8-bit 2's complement numbers.

More Examples

$$X = 00100111_{two}$$
$$= 2^5+2^2+2^1+2^0 = 32+4+2+1$$
$$= 39_{ten}$$

$$X = 11100110_{two}$$
$$-X = 00011010$$
$$= 2^4+2^3+2^1 = 16+8+2$$
$$= 26_{ten}$$
$$X = -26_{ten}$$

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
(K) 10	1024
(M) 20	1048576

Assuming 8-bit 2's complement numbers.

Converting Binary (2's C) to Decimal Practice

Convert binary 00011111 to decimal:

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
(K) 10	1024
(M) 20	1048576

Converting Decimal to Binary (2's C)

First Method: ***Division***

1. Divide by two – remainder is least significant bit.
2. Keep dividing by two until answer is zero, writing remainders from right to left.
3. Append a zero as the MS bit; if original number negative, take two's complement.

$X = 104_{ten}$

```
104/2 = 52 r0  bit 0
 52/2 = 26 r0  bit 1
 26/2 = 13 r0  bit 2
 13/2 = 6 r1   bit 3
  6/2 = 3 r0   bit 4
  3/2 = 1 r1   bit 5
  1/2 = 0 r1   bit 6
```

$X = 01101000_{two}$

Converting Decimal to Binary (2's C)

Second Method: ***Subtract Powers of Two***

1. Change to positive decimal number.
2. Subtract largest power of two less than or equal to number.
3. Put a one in the corresponding bit position.
4. Keep subtracting until result is zero.
5. Append a zero as MS bit; if original was negative, take two's complement.

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
(K) 10	1024
(M) 20	1048576

$X = 104_{ten}$

```
104 - 64 = 40   bit 6
 40 - 32 = 8    bit 5
  8 -  8 = 0    bit 3
```

$X = 01101000_{two}$

Converting Decimal to Binary Practice

Convert decimal 270 to binary using both methods described above:

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
(K) 10	1024
(M) 20	1048576

More Converting Decimal to Binary Practice

Convert decimal 255 to binary using both methods described above:

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
(K) 10	1024
(M) 20	1048576

Operations: Arithmetic and Logical

Recall: a data type includes *representation* and *operations*.

We now have a good representation for signed integers, so let's look at some ***arithmetic*** operations:

- Addition
- Subtraction
- Sign Extension

We'll also look overflow conditions for addition.

Multiplication, division, etc., can be built from these basic operations.

Logical operations are also useful:

- AND
- OR
- NOT

Addition

As we've discussed, 2's comp. addition is just binary addition.

- assume all integers have the same number of bits
- ignore carry out
- for now, assume that sum fits in n-bit 2's comp. representation

```
  01101000  (104)        11110110  (-10)
+ 11110000  (-16)     +            (-9)
  01011000  (88)                    (-19)
```

Assuming 8-bit 2's complement numbers.

Subtraction

Negate subtrahend (2nd no.) and add.

- assume all integers have the same number of bits
- ignore carry out
- for now, assume that difference fits in n-bit 2's comp. representation

01101000	(104)	11110110	(-10)
− 00010000	(16)	−	(-9)

is just

01101000	(104)	11110110	(-10)
+ 11110000	(-16)	+	(9)
01011000	(88)		(-1)

Assuming 8-bit 2's complement numbers.

Practice

Perform the Two's Complement operation to the following decimal numbers: - 56 - 14

Sign Extension

To add two numbers, we must represent them with the same number of bits.

If we just pad with zeroes on the left:

4-bit		8-bit	
`0100`	(4)	`00000100`	(still 4)
`1100`	(-4)	`00001100`	(12, not -4)

Instead, replicate the most significant bit -- the sign bit:

4-bit		8-bit	
`0100`	(4)	`00000100`	(still 4)
`1100`	(-4)	`11111100`	(still -4)

Overflow

If operands are too big,
then sum cannot be represented as an *n*-bit 2's comp number.

01000	(8)	11000	(-8)
+ 01001	(9)	+10111	(-9)
10001	(-15)	01111	(+15)

We have overflow if:

- signs of both operands are the same, and
- sign of sum is different.

Another test -- easy for hardware:

- carry into MS bit does not equal carry out

Logical Operations

Operations on logical TRUE or FALSE

- two states -- takes one bit to represent: TRUE=1, FALSE=0

View *n*-bit number as a collection of *n* logical values

- operation applied to each bit independently

A	B	A **AND** B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A **OR** B
0	0	0
0	1	1
1	0	1
1	1	1

A	**NOT** A
0	1
1	0

Examples of Logical Operations

AND

- useful for clearing bits
  - AND with zero = 0
  - AND with one = no change

```
      11000101
AND   00001111
      --------
      00000101
```

OR

- useful for setting bits
  - OR with zero = no change
  - OR with one = 1

```
      11000101
OR    00001111
      --------
      11001111
```

NOT

- unary operation -- one argument
- flips every bit

```
NOT   11000101
      --------
      00111010
```

Practice of Logical Operations

AND

- useful for clearing bits
  - AND with zero = 0
  - AND with one = no change

```
  10101010
AND 11100011
```

OR

- useful for setting bits
  - OR with zero = no change
  - OR with one = 1

```
  11001100
OR 00001111
```

NOT

- unary operation -- one argument
- flips every bit

```
NOT 10010110
```

Hexadecimal Notation

It is often convenient to write binary (base-2) numbers as hexadecimal (base-16) numbers instead.

- fewer digits -- four bits per hex digit
- less error prone -- easy to corrupt long string of 1's and 0's

Binary	Hex	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7

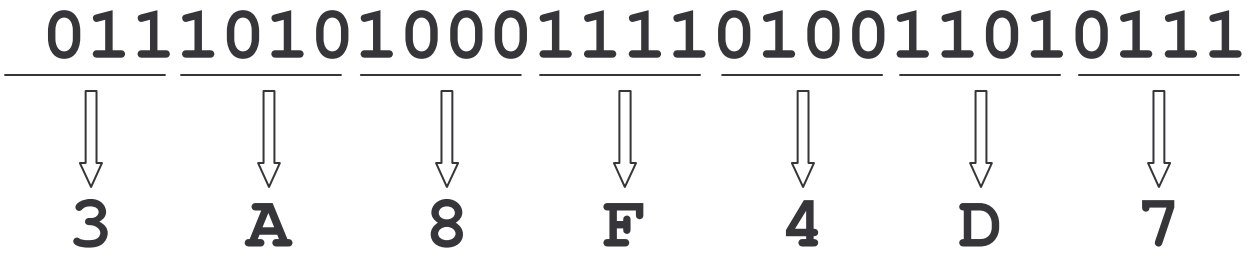
Binary	Hex	Decimal
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Memorize this table!!!!

Converting from Binary to Hexadecimal

Every four bits is a hex digit.

- start grouping from right-hand side

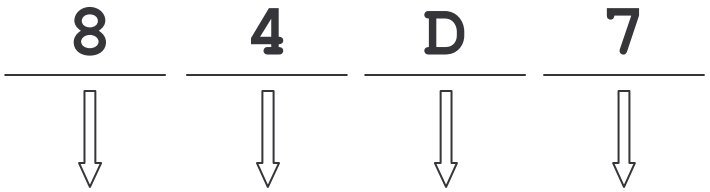


This is not a new machine representation, just a convenient way to write the number.

Converting from Hexadecimal to Decimal

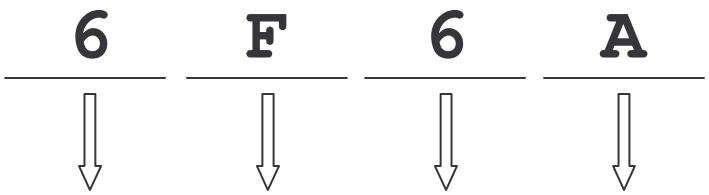
Every hex digit position has a base value

- multiply the value at the position by the base value



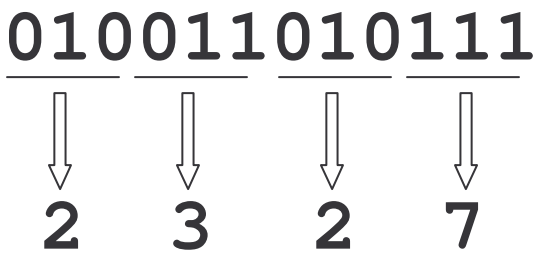
$$8x16^3 + 4x16^2 + 13x16^1 + 7x16^0 =$$
$$8x4096 + 4x256 + 13x16 + 7x1 =$$
$$32768 + 1024 + 208 + 7 =$$
$$34007$$

Practice Converting from Hex to Decimal

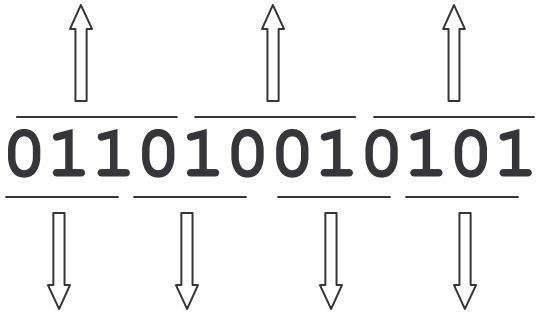


Octal

Octal is simply “base 8” number representation



Octal and Hex Practice



Multiplication by example

Consider $2_{10} \times 3_{10}$

multiplicand	0010	
multiplier	0011	
	0010	copy of multiplicand
	0010	copy of multiplicand
	0000	zero
	0000	zero
	00000110	

Text: ASCII Characters

ASCII: Maps 128 characters to 7-bit code.

– both printable and non-printable (ESC, DEL, …) characters

00	nul	10	dle	20	sp	30	0	40	@	50	P	60	`	70	p
01	soh	11	dc1	21	!	31	1	41	A	51	Q	61	a	71	q
02	stx	12	dc2	22	"	32	2	42	B	52	R	62	b	72	r
03	etx	13	dc3	23	#	33	3	43	C	53	S	63	c	73	s
04	eot	14	dc4	24	$	34	4	44	D	54	T	64	d	74	t
05	enq	15	nak	25	%	35	5	45	E	55	U	65	e	75	u
06	ack	16	syn	26	&	36	6	46	F	56	V	66	f	76	v
07	bel	17	etb	27	'	37	7	47	G	57	W	67	g	77	w
08	bs	18	can	28	(	38	8	48	H	58	X	68	h	78	x
09	ht	19	em	29	)	39	9	49	I	59	Y	69	i	79	y
0a	nl	1a	sub	2a	*	3a	:	4a	J	5a	Z	6a	j	7a	z
0b	vt	1b	esc	2b	+	3b	;	4b	K	5b	[	6b	k	7b	{
0c	np	1c	fs	2c	,	3c	<	4c	L	5c	\	6c	l	7c	|
0d	cr	1d	gs	2d	-	3d	=	4d	M	5d	]	6d	m	7d	}
0e	so	1e	rs	2e	.	3e	>	4e	N	5e	^	6e	n	7e	~
0f	si	1f	us	2f	/	3f	?	4f	O	5f	_	6f	o	7f	del

Data Communications

There was no standard for networks in the early days and as a result it was difficult for networks to communicate with each other.

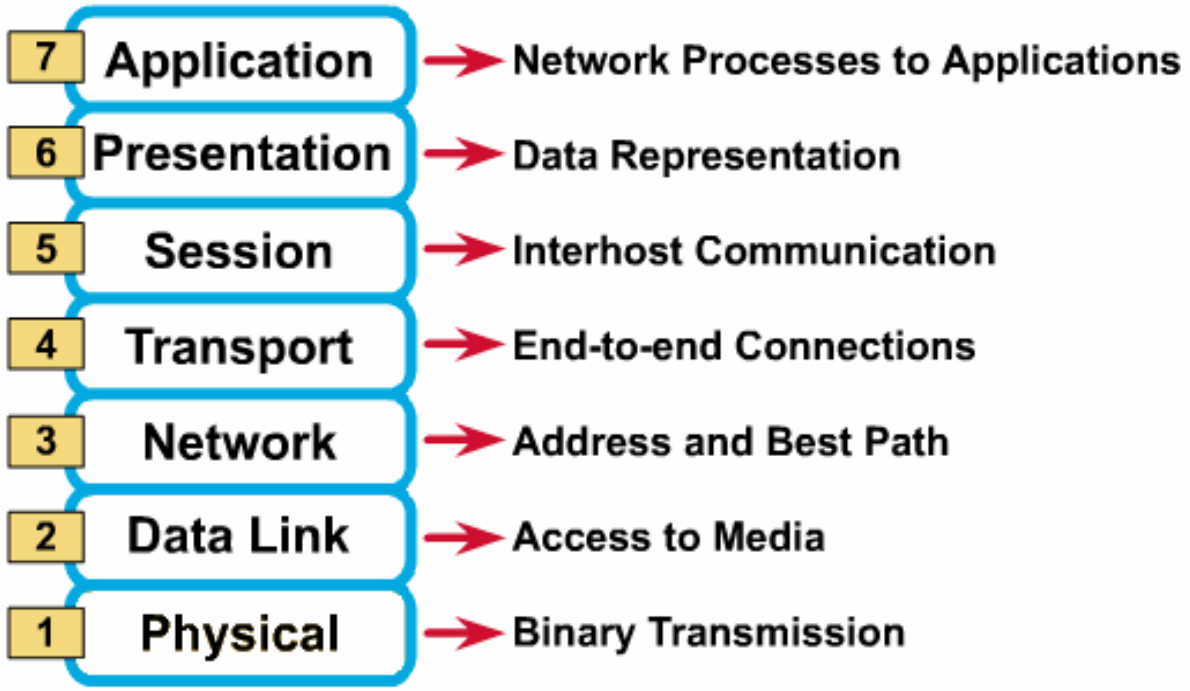
The International Organization for Standardization (ISO) recognized this and in 1984 introduced the Open Systems Interconnection (OSI) reference model.

The OSI reference model organizes network functions into seven numbered layers.

Each layer provides a service to the layer above it in the protocol specification and communicates with the same layer's software or hardware on other computers.

Layers 5-7 are concerned with services for the applications.

Layers 1-4 are concerned with the flow of data from end to end through the network



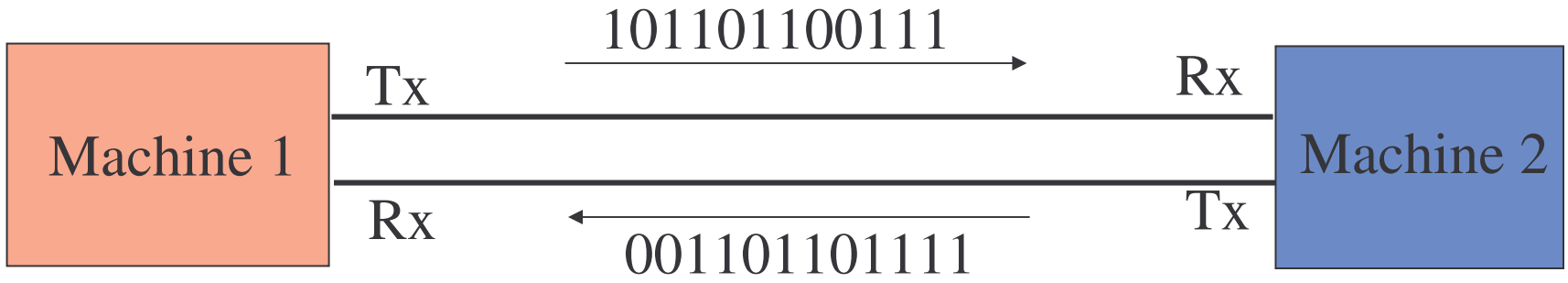
Physical Layer (1) – Serial Communications

The basic premise of serial communications is that one or two wires are used to transmit digital data.

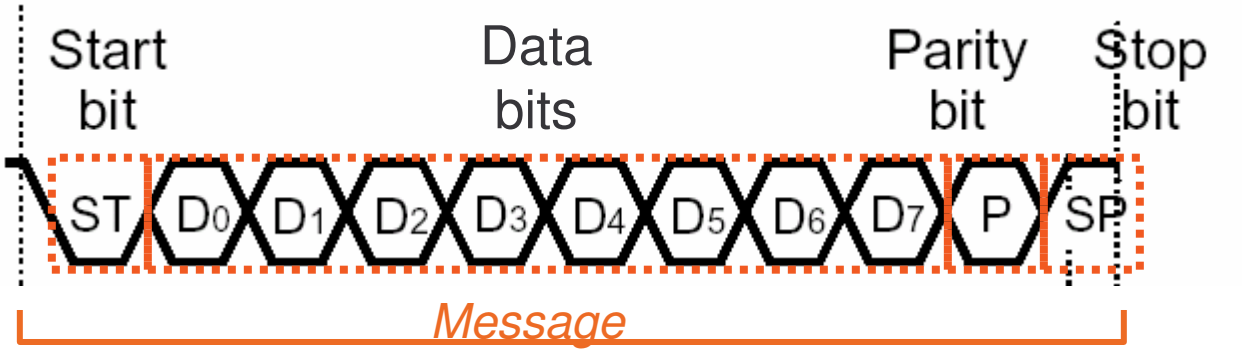
- Of course, ground reference is also needed (extra wire)

Can be one way or two way, usually two way, hence two communications wires.

Often other wires are used for other aspects of the communications (ground, “clear-to-send”, “data terminal ready”, etc).



Serial Communication Basics



Send one bit of the message at a time

Message fields

- Start bit (one bit)
- Data (LSB first or MSB, and size – 7, 8, 9 bits)
- Optional parity bit is used to make total number of ones in data even or odd
- Stop bit (one or two bits)

All devices on network or link must use same communications parameters

- The speed of communication must be the same as well (300, 600, 1200, 2400, 9600, 14400, 19200, etc.)

More sophisticated network protocols have more information in each message

- Medium access control – when multiple nodes are on bus, they must arbitrate for permission to transmit
- Addressing information – for which node is this message intended?
- Larger data payload
- Stronger error detection or error correction information
- Request for immediate response (“in-frame”)

Serial Communication Basics

RS232: rules on connector, signals/pins, voltage levels, handshaking, etc.

RS232: Fulfilling All Your Communication Needs, Robert Ashby

Quick Reference for RS485, RS422, RS232 and RS423

Not so quick reference:
The RS232 Standard: A Tutorial with Signal Names and Definitions,
Christopher E. Strangio

Bit vs Baud rates:
http://www.totse.com/en/technology/telecommunications/bits.html