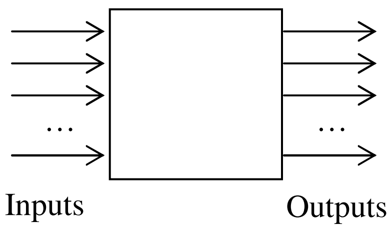
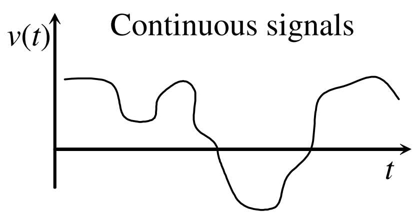
Ch. 1: Logic Design and Number Systems

1.1: Digital Logic Circuit Design

Two general types of circuits:

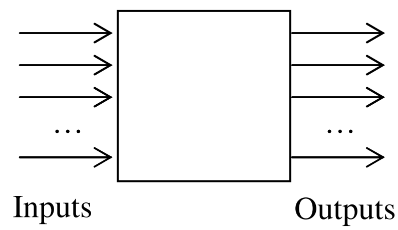
Analog Circuit



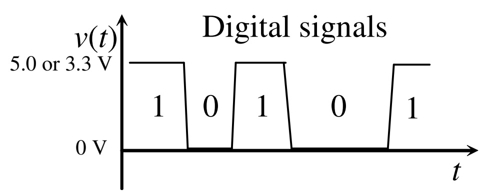


Inputs and outputs can only take on any value (2.5V, 1.2 V, 9.25V, etc)

Digital Circuit



A digital circuit manipulates information that has been encoded into ones and zeros.



Inputs and outputs can only take on only two unique values ("1" or "0")

1.2: Number Systems

- Decimal, or Base-10: Uses 10 symbols (digits 0,1,2,3,4,5,6,7,8,9) to represent numbers. Examples: 145 90 1098 or $(145)_{10}$ $(90)_{10}$ $(1098)_{10}$

- Binary, or Base-2: Used in digital logic (i.e., computers). Uses 2 symbols (binary digits, or **bits** 0,1) to represent numbers, letters, and everything else.
  Examples: $(110101)_2$ $(11101111)_2$ $(1010\ 1011\ 1000\ 1010)_2$
  Spaces added to improve readability

- Octal, or Base-8: Uses 8 symbols (0,1,2,3,4,5,6,7) to represent numbers.
  Examples: $(31670)_8$ $(531)_8$

- Hexadecimal, or Base-16: Uses 16 symbols (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F) to represent numbers.
  Examples: $(3F9A)_{16}$ $(E29F5B7AD4)_{16}$

Other bases possible (Base-3, Base-5, Base-9, Base-20, etc). Rarely used!

First 16 integers in Decimal, Binary and Hexadecimal. [Must learn!]

Decimal (base 10)	**Binary (base 2)**	**Hexadecimal (base 16)**
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

NOTE: 4 bits are needed to represent decimal 0 to 15 in binary system.

Some definitions:

Bit (b) = Binary Digit: a **1** or **0**.

Byte (B) = 8 bits. Basic unit of information storage.

1 K (kilo) = 2^{10} = 1,024 ~10^3 (~1 thousand, but not exactly)

1 M (mega) = 2^{20} = 1,048,576 ~10^6 (~1 million, but not exactly)

1 G (giga) = 2^{30} = 1,073,741,824 ~10^9 (~1 billion, but not exactly)

1 T (tera) = 2^{40} = 1,099,511,627,776 ~10^{12} (~1 trillion, but not exactly)

4 GB = 4 x 2^{30} bytes $\cong$ 4 billion bytes (but not exactly)

Number Base Conversions

How to convert a number from one base to another?

From <u>Any</u> Base to Base-10	**From Base-10 to <u>Other</u> Base**
Positional Weight Expansion	*Successive Division* for <u>Integer</u> part
	Successive Multiplication for <u>Fractional</u> part

Where $r \neq 10$

Positional Weight Expansion: Each "position" has a pre-assigned weight.

$(7642.74)_{10} =$

Base Conversion From Any Base to Base-10

From Any Base to Base-10 → Use *Positional Weight Expansion*

$(101111.101)_2 \rightarrow (\ ?\ )_{10}$

$(11\ 1001\ 0110)_2 \rightarrow (\ ?\ )_{10}$ [Spaces added to improve readability]

$(1023.12)_5 \rightarrow (\ ?\ )_{10}$

$(D43F.A)_{16} \rightarrow (\ ?\ )_{10}$

$(0.101)_2 \rightarrow (\ ?\ )_{10}$

$(0.A5)_{16} \rightarrow (\ ?\ )_{10}$

Base Conversion From Base-10 to Some Other Base

From Any Base to Base-10 → Use *Successive Division / Successive Multiplication*

$(24)_{10} \rightarrow (\ ?\ )_2$

$(136)_{10} \rightarrow (\ ?\ )_2$

$(746)_{10} \rightarrow (\ ?\ )_8$

$(746)_{10} \rightarrow (\ ?\ )_{16}$

$(0.625)_{10} \rightarrow (\ ?\ )_2$

$(0.625)_{10} \rightarrow (\ ?\ )_{16}$

$(0.3)_{10} \rightarrow (\ ?\ )_2$

$(24.625)_{10} \rightarrow (\ ?\ )_2$

What if you want to convert from Base-2 to Base-16? [Neither one is base-10]

1) Convert to base-10 first, then convert to base-16 [2-step process]
2) For "related" bases (ie. 2 and 16 are related by a power exponent: $\mathbf{2}^4=\mathbf{16}$),
   make groups of *n* bits, then convert each group to base-16.

$(10\ 1100\ 0110.1111\ 101)_2 \rightarrow (\ ?\ )_{16}$

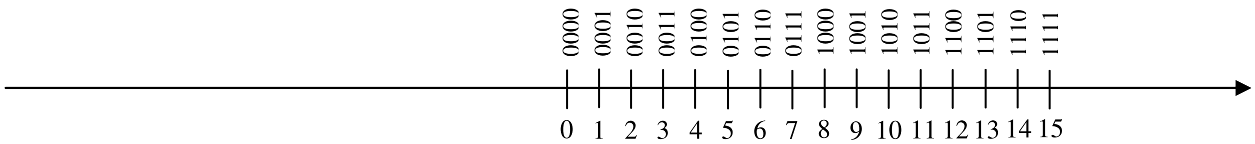
$(306.D)_{16} \rightarrow (\ ?\ )_2$

1.2.3: Unsigned and Signed Numbers

Unsigned Numbers

So far, we have dealt with ***unsigned*** numbers (positive only): 0, 1, 2, 3, 4, 5 …
If using 4 bits, can have 2^4=16 combinations.

0 = 0000	3 = 0011	6 = 0110	9 = 1001	:
1 = 0001	4 = 0100	7 = 0111	10 = 1010	:
2 = 0010	5 = 0101	8 = 1000	11 = 1011	15 = 1111



Using n bits $\Rightarrow$ Range = $0 \le N \le (2^n - 1)$ 2^n combinations

Largest unsigned number using n bits is $N_{max} = (2^n - 1)$. Ex: using 10 bits, can only represent ______ to ______.

How many bits (n) are needed to represent an unsigned number N?

$$0 \le N \le (2^n - 1) \Rightarrow n \ge \log_2(N+1) \qquad \text{[round up to whole number]}$$

Ex. How many bits needed to represent the unsigned number: 15? 2,000?

Signed Numbers

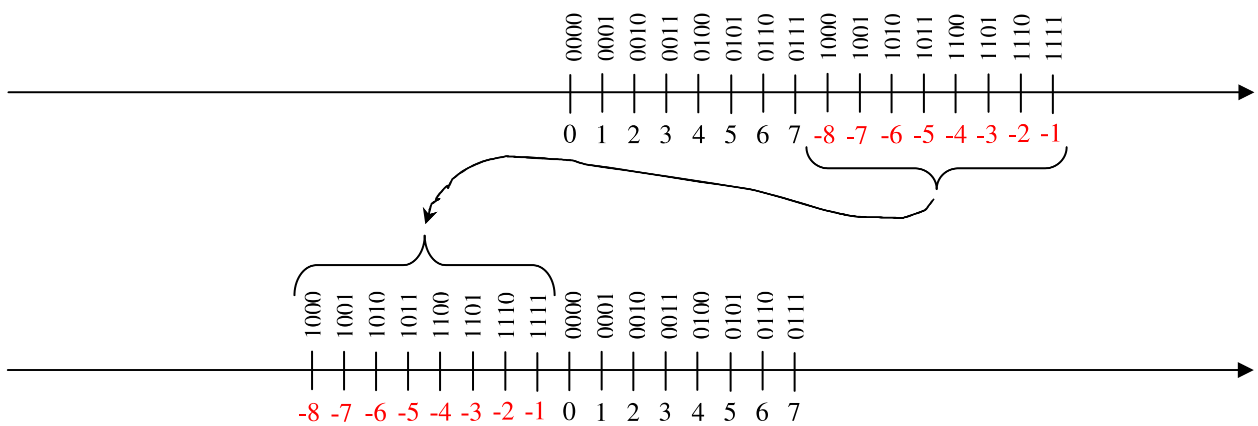
For binary arithmetic, we need to represent ***signed*** numbers (positive and negative numbers). Two ways to represent signed numbers using n bits:

1) Signed Magnitude Convention (Rarely used in practice) → 1 bit for sign, and remaining (n-1) bits represents magnitude.
   Ex. Use 4 bits to represent +5 and –5 in <u>signed magnitude</u> convention

$$\text{Using } n \text{ bits} \Rightarrow \text{Range} = -(2^{(n-1)} - 1) \le N \le +(2^{(n-1)} - 1)$$

NOTE: 2^n combinations

2) Two's Complement Convention (Most often used in practice) → Uses half of the combinations to represent negative numbers, by "wrapping around."



Half of combinations represent negative numbers, half represent positive numbers (zero included).

- If positive number: treat no differently as an unsigned (positive) number
- If negative number: represent as $2^n - |N|$ in binary

Consequence: Positive numbers start with **0**. Negative numbers start with **1**.

$$\text{Using } n \text{ bits} \Rightarrow \text{Range} = -(2^{(n-1)}) \le N \le +(2^{(n-1)} - 1)$$

NOTE: 2^n combinations

Ex. Use 4 bits to represent +5 and –5 in <u>two's complement</u> convention.

Ex. If two's complement convention with 16 bits, can only represent ____________ to ____________.

How many bits (n) are needed to represent a signed number N?

If $N > 0 \Rightarrow n \geq \log_2(N+1) + 1$ [round up n to nearest integer]

If $N < 0 \Rightarrow n \geq \log_2(|N|) + 1$ [round up n to nearest integer]

Signed Binary Numbers (using $n = 4$ bits)

Decimal	Signed Magnitude	Two's Complement
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
+0	0000	0000
–0		-----
–1		
–2		
–3		
–4		
–5		
–6		
–7		
–8		

MSB: most-significant bit (leftmost bit)

NOTE:
MSB = 0 if binary number is positive
MSB = 1 if binary number is negative
(regardless of convention!)

Shortcut to find two's complement representation:
1. Find binary equivalent of magnitude
2. Invert each bit
3. Add 1

Ex. Write –5 in two's complement using 4 bits.

Ex. Write –24 in two's complement using: (a) 6 bits; (b) 8 bits; (c) 4 bits

Given a binary number represented in two's complement, one can obtain the decimal equivalent.

Ex: $(011000)_2 = (\ ?\ )_{10}$

Ex: $(101000)_2 = (\ ?\ )_{10}$

Ex: $(1110\ 1000)_2 = (\ ?\ )_{10}$

Summary: For two's complement number conversions:

	Positive Number	Negative Number
Decimal to Binary:	Successive Division	$2^n - \lvert x \rvert$, or shortcut (invert, add 1)
Binary to decimal:	Normal positional weight expansion	Modified positional weight expansion (1st weight is negative)

1.2.2, 1.2.4: Binary Arithmetic

1.2.2: Binary Addition. Given A and B, wish to compute $A + B$ in binary.

Rules of binary addition: (Adding 2 bits ⇒ 4 combinations)

A	0	0	1	1
B	+ 0	+ 1	+ 0	+ 1
	0	1	1	$\underline{\mathbf{1}}$ 0

Underlined: **carry bit** into the next higher column.

What about 3-bit addition? (8 combinations)

A	0	0	0	0	1	1	1	1
B	0	0	1	1	0	0	1	1
C	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1
	0	1	1	$\underline{\mathbf{1}}$ 0	1	$\underline{\mathbf{1}}$ 0	$\underline{\mathbf{1}}$ 0	$\underline{\mathbf{1}}$ 1

Can be extended to many more bits.

When performing binary addition, must restrict result to given number of bits.
Overflow occurs when result is out of range (for given number of bits).

Assume unsigned (positive) numbers in following examples.

Ex. Add 6+7 in binary (use 4 bits).

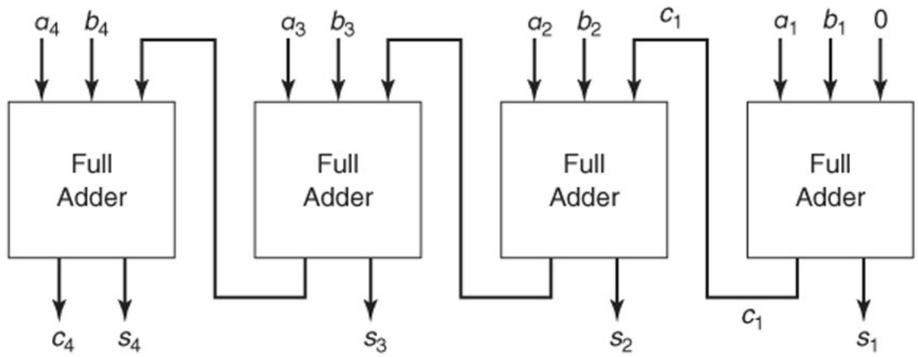
Ex. Add 13+5 in binary (use 4 bits).

Ex. Add 13+5 in binary (use 5 bits).

For unsigned numbers arithmetic:
How is *Overflow* detected?

Multi-bit addition is done with an "*adder*" circuit (we will learn this later).

Figure 1.2 A 4-bit adder.



1.2.4: Binary Subtraction. Given A and B, wish to compute $A - B$ in binary.
Key: $A - B = A + (-B)$. Convert the *subtraction* into an *addition* problem.

Ex. Compute 7–5 in binary.
(use 4 bits)

Ex. Compute (–5 + 7) in binary.
(use 4 bits)

Ex. Compute (–5 + 7) in binary.
(use 6 bits)

Ex. Compute (–5) – (6) in binary.
(use 4 bits)

For signed numbers arithmetic: How is *Overflow* detected?

When *Overflow* occurs ⇒ Binary result does not equal expected decimal result, because resulting number is *out of range*.

Must be told whether dealing with unsigned or signed numbers!

Ex. What is being computed here?

If unsigned		If signed 2's complement	
0100	4	0100	4
+ 1101	+ ___	+ 1101	+ ___

1.2.6, 1.2.7: Binary Encoding

Goal: Use binary combinations to represent distinct values or objects
Fact: A n-bit binary code can represent at most 2^n distinct objects

These codes are widely used:

- BCD (Binary Coded Decimal) Codes
- Gray Code
- ASCII (American Standard Code for Information Interchange) Code

- **BCD (Binary Coded Decimal) Codes**

- Use 4-bit binary combination to represent each **decimal** digit (0 through 9).

Table 1.7 Binary coded decimal codes. (Page 18)

Decimal digit	8421 code	5421 code	2421 code	Excess 3 code	2 of 5 code
0	0000	0000	0000	0011	11000
1	0001	0001	0001	0100	10100
2	0010	0010	0010	0101	10010
3	0011	0011	0011	0110	10001
4	0100	0100	0100	0111	01100
5	0101	1000	1011	1000	01010
6	0110	1001	1100	1001	01001
7	0111	1010	1101	1010	00110
8	1000	1011	1110	1011	00101
9	1001	1100	1111	1100	00011
unused	1010	0101	0101	0000	any of
Invalid	1011	0110	0110	0001	the 22
combinations	1100	0111	0111	0010	patterns
	1101	1101	1000	1101	with 0, 1,
	1110	1110	1001	1110	3, 4, or 5
	1111	1111	1010	1111	1's

Ex1. Represent the decimal number 739 using:

(i) 8421 code:

(ii) 5421 code:

(iii) XS3 code:

Ex2. What do the following represent in decimal?

(i) $(0011 \ \ 1001)_{\text{XS3 code}}$:

(ii) $(0000 \ \ 0101)_{\text{2421 code}}$:

(iii) $(0001 \ \ 0101)_2$:

- **Gray Code**

- Uses n bits to represent 2^n decimal digits (from 0 to $2^n - 1$)
- Special property: consecutive numbers differ in only one bit
- Useful in coding the position of a continuous device (i.e., a wheel)

Table 1.9 Gray code. $n = 4$ bits

(Page 20)

Number	Gray code	Number	Gray code
0	0000	8	1100
1	0001	9	1101
2	0011	10	1111
3	0010	11	1110
4	0110	12	1010
5	0111	13	1011
6	0101	14	1001
7	0100	15	1000

Ex3. Represent the decimal number 739 using 4-bit gray code.

$739 \Rightarrow (0100 \quad 0010 \quad 1101)_{\text{Gray Code}}$

• ASCII (American Standard Code for Information Interchange) Code

- Uses 7 bits to represent printable characters from standard keyboard
- Also represents 32 non-printable control codes (carriage return, SHIFT, CTRL)

American Standard Code for Information Interchange (ASCII) (Page 19)

		$a_6a_5a_4$							
Hex	$a_3a_2a_1a_0$	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0	**0000**	NUL	DLE	space	0	@	P	‘	p
1	**0001**	SOH	DC1	!	1	A	Q	a	q
2	**0010**	STX	DC2	“	2	B	R	b	r
3	**0011**	ETX	DC3	#	3	C	S	c	s
4	**0100**	EOT	DC4	$	4	D	T	d	t
5	**0101**	ENQ	NAK	%	5	E	U	e	u
6	**0110**	ACK	SYN	&	6	F	V	f	v
7	**0111**	BEL	ETB	‘	7	G	W	g	w
8	**1000**	BS	CAN	(	8	H	X	h	x
9	**1001**	HT	EM	)	9	I	Y	i	y
A	**1010**	LF	SUB	*	:	J	Z	j	z
B	**1011**	VT	ESC	+	;	K	[	k	{
C	**1100**	FF	FS	,	<	L	\	l	|
D	**1101**	CR	GS	–	=	M	]	m	}
E	**1110**	SO	RS	.	>	N	^	n	~
F	**1111**	SI	US	/	?	O	_	o	delete
	Hex →		1	2	3	4	5	6	7

Ex: ASCII for ***G*** is

$a_6\, a_5\, a_4\, a_3\, a_2\, a_1\, a_0$

1 0 0 0 1 1 1

100 0111 = $(47)_{hex}$

Ex: ASCII for ***m*** is ?

= ($)_{hex}$

Ex4. Code the string ***244 Logic*** using ASCII code. Convert to hex too.

2	4	4	space	L	o	g	i	c
011 0010	011 0100	011 0100	010 0000	010 1100	110 1111	110 0111	110 1001	110 0011
32	34	34	20	4C	6F	67	69	63

Ex5. Code the string ***4 + 5 = 9*** using ASCII code. Convert to hex too.

4	space	+	space	5	space	=	space	9
011 0100	010 0000	010 1011	010 0000	011 0101	010 0000	011 1101	010 0000	011 1001
34	20	2B	20	35	20	3D	20	39

Ex6. What does hexadecimal ASCII string “45 78 61 6D 23 31” represent?

45 78 61 6D 23 31