ECNG1014 - Digital Electronics

Lecture 1a

Number Systems (Part 1)

Marcus L. George

Overview

- Understanding decimal numbers
- Binary, Octal and Hexadecimal numbers
  - The basis of computers!
- Conversion between different number systems

Digital Computer Systems

- Digital systems consider *discrete* amounts of data.
  - Examples
    - 26 letters in the alphabet
    - 10 decimal digits
- Larger quantities can be built from discrete values:
  - Words made of letters (e.g.. Apples are Red)
  - Numbers made of decimal digits (e.g.. 239875.32)
- Computers operate on *binary* values (0 and 1)
- Easy to represent binary values electrically
  - Can be implemented using circuits
  - Create the building blocks of modern computers

Understanding Decimal Numbers

- Decimal numbers are made of decimal digits: (0,1,2,3,4,5,6,7,8,9)
- But how many items does a decimal number represent?
  - $8653 = 8x10^3 + 6x10^2 + 5x10^1 + 3x10^0$
- What about fractions?
  - $97654.35 = 9x10^4 + 7x10^3 + 6x10^2 + 5x10^1 + 4x10^0 + 3x10^{-1} + 5x10^{-2}$
  - In formal notation -> $(97654.35)_{10}$

Understanding Binary Numbers

- Binary numbers are made of binary digits (bits):
  - 0 and 1
- How many items does a binary number represent?
  - $(1011)_2 = 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0 = (11)_{10}$
- What about fractions?
  - $(110.10)_2 = 1x2^2 + 1x2^1 + 0x2^0 + 1x2^{-1} + 0x2^{-2}$
- Groups of eight bits are called a *byte*
  - $(11001001)_2$
- Groups of four bits are called a *nibble*.
  - $(1101)_2$

Why Use Binary Numbers?

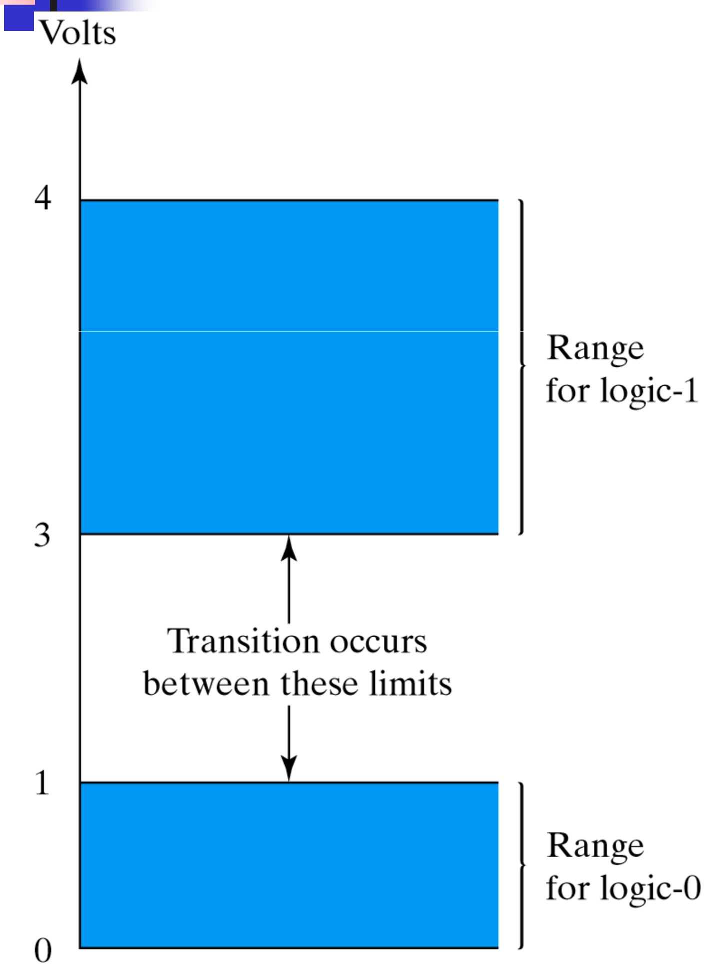
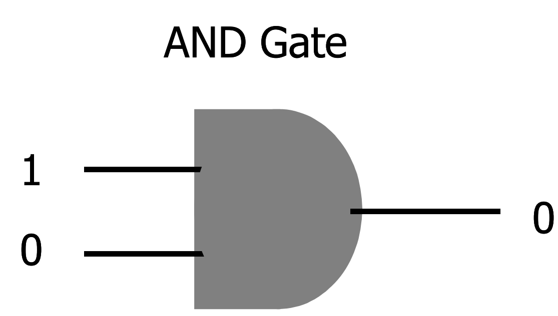


Fig. 1-3 Example of binary signals

- Easy to represent 0 and 1 using electrical values.
- Possible to tolerate noise.
- Easy to transmit data
- Easy to build binary circuits.



Understanding Octal Numbers

- Octal numbers are made of octal digits: (0,1,2,3,4,5,6,7)
- Octal numbers don't use digits 8 or 9
- How many items does an octal number represent?
  - $(4536)_8 = 4x8^3 + 5x8^2 + 3x8^1 + 6x8^0 = (2398)_{10}$
- What about fractions?
  - $(465.27)_8 = 4x8^2 + 6x8^1 + 5x8^0 + 2x8^{-1} + 7x8^{-2}$

Understanding Hexadecimal Numbers

- Hexadecimal numbers are made of <u>16</u> digits:
  - (0,1,2,3,4,5,6,7,8,9,A, B, C, D, E, F)
- How many items does an hex number represent?
  - $(3A9F)_{16} = 3x16^3 + 10x16^2 + 9x16^1 + 15x16^0 = 14999_{10}$
- What about fractions?
  - $(2D3.5)_{16} = 2x16^2 + 13x16^1 + 3x16^0 + 5x16^{-1} = 723.3125_{10}$
- Note that *each* hexadecimal digit can be represented with four bits.
  - $(1110)_2 = (E)_{16}$
- Groups of four bits are called a *nibble.*
  - $(1110)_2$

Putting It All Together

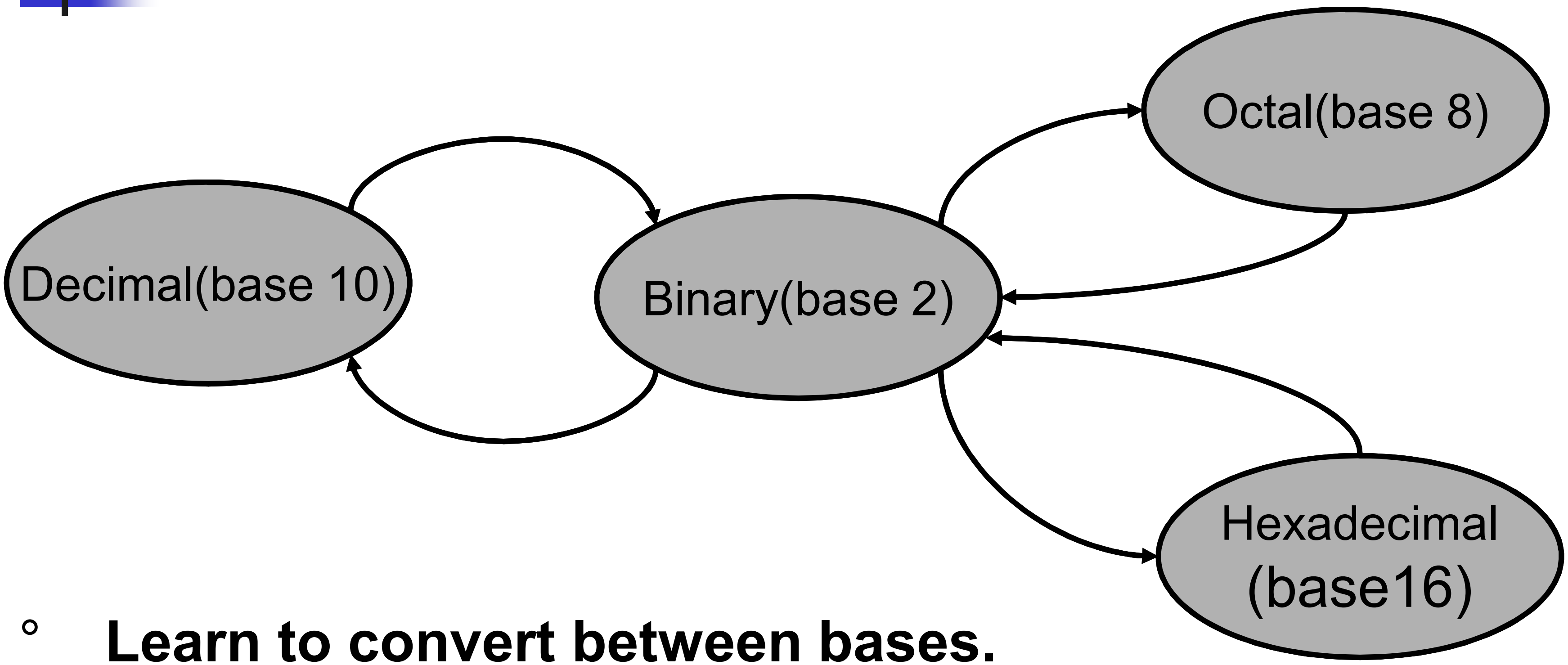
Decimal	Binary	Octal	Hexadecimal	BCD
0	0	0	0	0000
1	1	1	1	0001
2	10	2	2	0010
3	11	3	3	0011
4	100	4	4	0100
5	101	5	5	0101
6	110	6	6	0110
7	111	7	7	0111
8	1000	10	8	1000
9	1001	11	9	1001
10	1010	12	A	0001 0000
11	1011	13	B	0001 0001
12	1100	14	C	0001 0010
13	1101	15	D	0001 0011
14	1110	16	E	0001 0100
15	1111	17	F	0001 0101

Binary, octal, and hexadecimal similar

Easy to build circuits to operate on these representations

Possible to convert between the three formats

Conversion Between Number Bases



- Learn to convert between bases.
- Already demonstrated how to convert from binary to decimal.

Convert an Integer *from* Decimal *to* Another Base

For each digit position:

1. **Divide decimal number by the base (e.g. 2)**
2. **The remainder is the lowest-order digit**
3. **Repeat first two steps until no divisor remains.**

Example for $(13)_{10}$:

	Integer Quotient		Remainder	Coefficient
13/2 =	6	+	½	$a_0 = 1$
6/2 =	3	+	0	$a_1 = 0$
3/2 =	1	+	½	$a_2 = 1$
1/2 =	0	+	½	$a_3 = 1$

Answer $(13)_{10} = (a_3\, a_2\, a_1\, a_0)_2 = (1101)_2$

Convert a Fraction *from* Decimal *to* Another Base

For each digit position:

1. **Multiply decimal number by the base (e.g. 2)**
2. **The integer is the highest-order digit**
3. **Repeat first two steps until fraction becomes zero.**

Example for $(0.625)_{10}$:

	Integer		Fraction	Coefficient
0.625 x 2 =	1	+	0.25	$a_{-1} = 1$
0.250 x 2 =	0	+	0.50	$a_{-2} = 0$
0.500 x 2 =	1	+	0	$a_{-3} = 1$

Answer $(0.625)_{10} = (0.a_{-1}\, a_{-2}\, a_{-3})_2 = (0.101)_2$

Convert an Integer *from* Decimal *to* Octal

For each digit position:

1. **Divide decimal number by the base (8)**
2. **The *remainder* is the lowest-order digit**
3. **Repeat first two steps until no *divisor* remains.**

Example for $(175)_{10}$:

	Integer Quotient		Remainder	Coefficient
175/8 =	21	+	7/8	$a_0 = 7$
21/8 =	2	+	5/8	$a_1 = 5$
2/8 =	0	+	2/8	$a_2 = 2$

Answer $(175)_{10} = (a_2\, a_1\, a_0)_2 = (257)_8$

Convert an Fraction *from* Decimal *to* Octal

For each digit position:

1. **Multiply decimal number by the base (e.g. 8)**
2. **The *integer* is the highest-order digit**
3. **Repeat first two steps until fraction becomes zero.**

Example for $(0.3125)_{10}$:

	Integer		Fraction	Coefficient
0.3125 x 8 =	2	+	0.5	$a_{-1} = 2$
0.5000 x 8 =	4	+	0	$a_{-2} = 4$

Answer $(0.3125)_{10} = (0.24)_8$

Converting Between Base 16 and Base 2

- Conversion is easy!
  - Determine 4-bit value for each hex digit
- Note that there are16 different values of four bits
- Easier to read and write in hexadecimal.
- Representations are equivalent!

$$3A9F_{16} = \underbrace{0011}_{3}\ \underbrace{1010}_{A}\ \underbrace{1001}_{9}\ \underbrace{1111}_{F}{}_2$$

Converting Between Base 16 and Base 8

- Convert from Base 16 to Base 2
- Regroup bits into groups of three starting from right
- Ignore leading zeros
- Each group of three bits forms an octal digit.

$$3A9F_{16} = \underbrace{0011}_{3}\ \underbrace{1010}_{A}\ \underbrace{1001}_{9}\ \underbrace{1111}_{F}{}_2$$

$$\downarrow$$

$$35237_8 = \underbrace{011}_{3}\ \underbrace{101}_{5}\ \underbrace{010}_{2}\ \underbrace{011}_{3}\ \underbrace{111}_{7}{}_2$$

Can you figure out the equivalent base 4 representation?????

Shortcut Method to converting decimal to binary (Integers)

2^n	Decimal Representation (base 10)	Binary Representation (base 2)
2^0	1	1
2^1	2	10
2^2	4	100
2^3	8	1000
2^4	16	10000
2^5	32	100000
2^6	64	1000000

Shortcut Method to converting decimal to binary (Fractions)

2^n	Decimal Representation (base 10)	Binary Representation (base 2)
2^{-1}	0.5	0.1
2^{-2}	0.25	0.01
2^{-3}	0.125	0.001
2^{-4}	0.0626	0.0001
2^{-5}	0.03125	0.00001

Shortcut Method to converting decimal to binary (Example)

eg. $(39.75)_{10} = (32)_{10} + (4)_{10} + (2)_{10} + (1)_{10} + (0.5)_{10} + (0.25)_{10}$

$= (100000)_2 + (100)_2 + (10)_2 + (1)_2 + (0.1)_2 + (0.01)_2$

$= (100111.11)_2$

ECNG1014 - Digital Electronics

Lecture 1b

Number Systems (Part 2)

Marcus L. George

Overview

- Binary Addition, Subtraction, Multiplication
- Value ranges of numbers
- Representing positive and negative numbers
- Creating the complement of a number
  - Make a positive number negative (and vice versa)

Binary Addition

- Binary addition is very simple.
- This is best shown in an example of adding two binary numbers…

```
  1   1   1   1    1   1  <------------ carries
      1   1   1   1   0   1
+         1   0   1   1   1
---------------------
  1   0   1   0   1   0   0
```

Binary Subtraction

- We can also perform subtraction (with borrows in place of carries).
- Let's subtract (10111)2 from (1001101)2…

```
             1           10              <------------ borrows
          0  10  10   0   0  10

          1   0   0   1   1   0   1
  -               1   0   1   1   1
  ---------------------------------
              1   1   0   1   1   0
```

Binary Multiplication

- Binary multiplication is much the same as decimal multiplication, except that the multiplication operations are much simpler…

```
              1  0  1  1  1
x                1  0  1  0
-----------------------------
              0  0  0  0  0
           1  0  1  1  1
        0  0  0  0  0
     1  0  1  1  1
-----------------------------
     1  1  1  0  0  1  1  0
```

How To Represent Signed Numbers

- Plus and minus sign used for decimal numbers: 25 (or +25), -16, etc.
- For computers, desirable to represent everything as *bits*.
- Three types of signed binary number representations:
  - signed magnitude
  - 1's complement
  - 2's complement.
- In each case: left-most bit indicates sign: positive (0) or negative (1).

Signed Magnitude Representation

Consider ***signed magnitude***:

$0\underline{0001100}_2 = 12_{10}$

Sign bit Magnitude

$1\underline{0001100}_2 = -12_{10}$

Sign bit Magnitude

This demonstrates 8-bit signed magnitude

4-bit Signed Magnitude Representation

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

One’s Complement Representation

- The one’s complement of a binary number involves inverting all bits.
  - 1’s comp of 00110011 is 11001100
  - 1’s comp of 10101010 is 01010101
- For an n bit number **N** the 1’s complement is (2n-1) – N.
- To find negative of 1’s complement number take the 1’s complement.

One's Complement Representation (cont.)

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

Two's Complement Representation

- The two's complement of a binary number involves inverting all bits and adding 1.
  - 2's comp of 00110011 is 11001101
  - 2's comp of 10101010 is 01010110
- For an n bit number **N** the 2's complement is (2n-1) – N + 1.
- To find negative of 2's complement number take the 2's complement.

Two's Complement Representation (cont.)

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

2’s Complement Subtraction

- Using 2’s complement numbers, follow steps for subtraction
- For example, suppose we wish to subtract +(0001)2 from +(1100)2.
- Let’s compute (12)10 - (1)10.
  - (12)10 = +(1100)2 = (01100)2 in 2’s comp.
  - (-1)10 = -(0001)2= (11111)2 in 2’s comp.

```
                 0 1 1 0 0
               - 0 0 0 0 1
               -------------
2’s comp         0 1 1 0 0
Add            + 1 1 1 1 1
               -------------
Final Result   1 0 1 0 1 1
```

Ignore Carry

Step 1: Take 2’s complement of 2nd operand

Step 2: Add binary numbers

Step 3: Ignore carry bit

ECNG1014 - Digital Electronics

Lecture 1c

Number Systems (Part 3)

Marcus L. George

Overview

- Fixed Point Numbers
- Floating Point Numbers
- Binary Coded Decimal (BCD)
- Packed and Unpacked BCD Format
- ASCII and Gray Code

Fixed Point Numbers

- In fixed point numbers, the position of the decimal point remains fixed.
  - Place 12.875 in 8-bit fixed for format with 4 bits before and after decimal point
    - Ans=11001110 (easily understood as 1100.1110)

Floating Point Numbers

- In floating point numbers, the position of the decimal point remains is variable depending on number to be represented.
- Floating point numbers consists of 3 parts:
  - Sign Bit
  - Biased Exponent
  - Mantissa
- Two floating point representations:
  - Single Precision (32 bits)
  - Double Precision (64 bits)
- These are known as the IEEE 754 standards for 32-bit and 64-bit numbers respectively

IEEE 754 floating-point standard for 32-bit numbers

- Single Precision Representation:
  - Sign Bit → 1 bit (bit 31)
  - Biased Exponent → 8 bits (bits 23 - 30)
  - Mantissa → 23 bits (bits 0 - 22)
  - Bias = 127

IEEE 754 floating-point standard for 64-bit numbers

- Double Precision Representation:
  - Sign Bit → 1 bit (bit 63)
  - Biased Exponent → 11 bits (bits 52 - 62)
  - Mantissa → 52 bits (bits 0 - 51)
  - Bias = 1023

Converting to 32-bit Single Precision Floating Point Format

e.g. Convert 23.875 to 32-bit F.P. single precision

- Step 1: Convert number to binary

  23.875 → 10111.111 in binary

- Step 2: Convert to standard form

  10111.111 → 1.0111111 × 2^4

- Step 3: Determine Sign Bit

  Number is clearly positive, hence sign bit = 0

Converting to 32-bit Single Precision Floating Point Format (Cont.)

- Step 4: Compute Biased Exponent
  - We obtain the exponent from the number in standard form. i.e. exponent = 4 = 100
  - We previously stated that the bias = 127, hence biased exponent = exponent + bias

$$= 100 + 01111111$$

$$= 10000011$$

Converting to 32-bit Single Precision Floating Point Format (Cont.)

- Step 5: Determine Mantissa
  - Use everything after decimal point and pad up with zeros
  - In 1.0111111 × 2^4, 0111111 lies after decimal point
  - Hence Mantissa is 01111110000000000000000

Converting to 32-bit Single Precision Floating Point Format (Cont.)

- Step 6: Put all components together
  - Sign Bit = 0
  - Biased Exponent = 10000011
  - Mantissa = 01111110000000000000000

  Hence the 23.875 in floating point format is:

  0 10000011 01111110000000000000000
- Homework: Convert 23.875 to 64-bit F.P. double precision format

Binary Coded Decimal (BCD)

- Binary coded decimal (BCD) represents each decimal digit with four bits
  - e.g. 0011 0010 1001 = 329_{10}
  - This is ***NOT*** the same as 001100101001_2
- BCD not very efficient
- Easier to read?
- Used to encode numbers for seven-segment displays (will see in lab 3)

Binary Coded Decimal (BCD)

Decimal	Binary	Octal	Hexadecimal	BCD
0	0	0	0	0000
1	1	1	1	0001
2	10	2	2	0010
3	11	3	3	0011
4	100	4	4	0100
5	101	5	5	0101
6	110	6	6	0110
7	111	7	7	0111
8	1000	10	8	1000
9	1001	11	9	1001
10	1010	12	A	0001 0000
11	1011	13	B	0001 0001
12	1100	14	C	0001 0010
13	1101	15	D	0001 0011
14	1110	16	E	0001 0100
15	1111	17	F	0001 0101

Packed and Unpacked BCD

- Packed BCD format stores 2 BCD in 1 byte
  - e.g. 59 → 01011001
- Unpacked BCD format stores only 1 BCD in 1 byte
  - e.g. 59 → 00000101 00001001

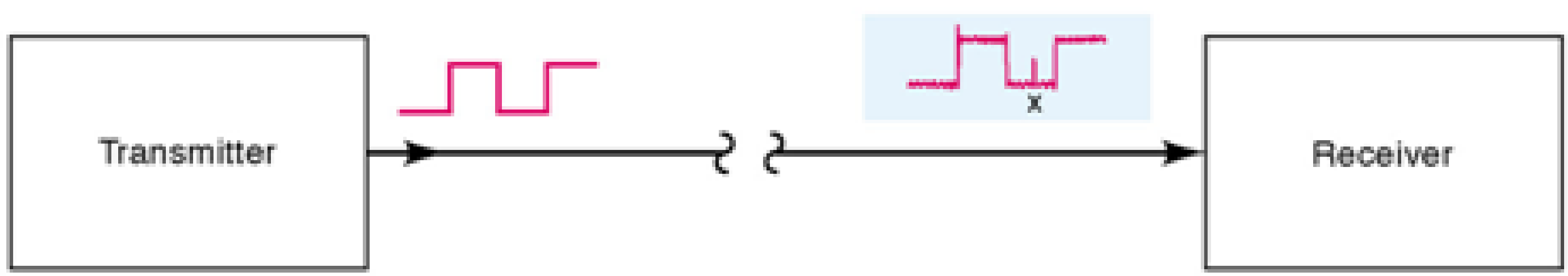
ASCII

- American Standard Code for Information Interchange
- ASCII is a 7-bit code, frequently used with an 8th bit for error detection (parity bit)

Character	ASCII (bin)	ASCII (hex)	Decimal	Octal
A	1000001	41	65	101
B	1000010	42	66	102
C	1000011	43	67	103
...				
Z				

ASCII Codes and Data Transmission

- ASCII Codes
  - A – Z (26 codes), a – z (26 codes)
  - 0-9 (10 codes), others (@#$%^&*….)
- Transmission susceptible to noise
- Typical transmission rates (1500 Kbps, 56.6 Kbps)
  - How to keep data transmission accurate?



ASCII Codes and Data Transmission

- Error detection done using an additional bit called a Parity bit
- Parity codes are formed by concatenating a *parity bit, P* to each code word of C.
- In an *odd-parity code*, the parity bit is specified so that the total number of ones is odd.
- In an even-*parity code*, the parity bit is specified so that the total number of ones is even.

ASCII Codes and Data Transmission

P	Information Bits

`1 1 0 0 0 0 1 1`
↑
Added even parity bit

`0 1 0 0 0 0 1 1`
↑
Added odd parity bit

Gray Code

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

- Gray code is not a number system.
  - It is an alternate way to represent four bit data
- Only one bit changes from one decimal digit to the next

Next Topic:

-->Boolean Algebra