Dept. of Computer Science and Engineering
University of Rajshahi
www.ru.ac.bd

Dr. Shamim Ahmad

Number Systems

- Decimal (Base 10)
  - 10 digits (0,1,2,3,4,5,6,7,8,9)
- Binary (Base 2)
  - 2 digits (0,1)
    - Digits are often called bits (binary digits)
- Hexadecimal (Base 16)
  - 16 digits (0-9,A,B,C,D,E,F)
    - Often referred to as Hex

Number Systems

Octal and Hexadecimal Numbers

- Octal = Radix 8
- Only eight digits: 0 to 7
- Digits 8 and 9 not used
- Hexadecimal = Radix 16
- 16 digits: 0 to 9, A to F
- A=10, B=11, …, F=15
- First 16 decimal values (0 to15) and their values in binary, octal and hex. Memorize table

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

Positional Notation

- Each digit is weighted by the base(r) to the positional power
- $N = d_{n-1}d_{n-2}\ldots d_0.d_1d_2\ldots d_m$
  $= (d_{n-1} \times r^{n-1}) + (d_{n-2} \times r^{n-1}) + \ldots + (d_{0\,1} \times r^0) +$
  $(d_1 \times r^1) + (d_2 \times r^2) + \ldots (d_m \times r^m)$

- Example : 872.64_{10}
  $(8 \times 10^2) + (7 \times 10^1) + (2 \times 10^0)$
  $+ (6 \times 10^{-1}) + (4 \times 10^{-2})$
- Example: 1011.1_2 = ?
- Example :$12A_{16}$ = ?

Positional Notation
(Solutions to Example Problems)

- $872.64_{10} = 8x10^2 + 7x10^1 + 2x10^0 + 6x10^{-1} + 4\ x10^{-2}$

  800 + 70 + 2 + .6 + .04

Positional Notation
(Solutions to Example Problems)

- 1011.1_2

  $= 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0 + 1x2^{-1}$

  = 8 + 0 + 2 + 1 + .5

  $= 11.5_{10}$

Positional Notation
(Solutions to Example Problems)

- $12A_{16} = 1x16^2 + 2x16^1 + Ax16^0$

  = 256 + 32 + 10

  $= 298_{10}$

Powers of Bases

2^{-3}= .125
2^{-2}= .25
2^{-1}= .5
$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1024$
$2^{11} = 2048$
$2^{12} = 4096$

$16^0 = 1$
$16^1 = 16 = 2^4$
$16^2 = 256 = 2^8$
$16^3 = 4096 = 2^{12}$

2^{10} = 1024 = 1Kb
2^{20} = 1,048,576 = 1Mb
2^{30} = 1,073,741,824 = 1Gb

Determining What Base is being Used

- Subscripts
  - 874_{10} 1011_2 $AB9_{16}$
  - $AB9_{(16)}$
- Prefix Symbols
  - (None) 874 %1011 $AB9
- Postfix Symbols
  - AB9H
- If I am only working with one base there is no need to add a symbol.

Conversion from Base R to Decimal

- Use Positional Notation
- %11011011 = $?_{10}$
- $3A94 = $?_{10}$

Conversion from Binary to Decimal

- Use Positional Notation
- %11011011 = $?_{10}$
  %11011011
  $= 1x2^7 + 1x2^6 + 1x2^4 + 1x2^3 + 1x2^1 + 1x2^0$
  = 128 + 64 + 16 + 8 + 2 + 1
  $= 219_{10}$

Conversion from Hexadecimal to Decimal

- $3A94 = $?_{10}$
- $3A94 = $3x16^3 + Ax16^2 + 9x16^1 + 4x16^0$
  =12288 + 2560 + 144 + 4
  =15996

Conversion from Decimal to Base R

- Use Successive Division
  - Repeatedly divide by the desired base until 0 is reached
  - Save the remainders as the final answer
  - The first remainder is the LSB (least significant bit); the last remainder is the MSB (Most significant bit)
- $437_{10} = ?_2$
  $= 110110101_2$
- $437_{10} = ?_{16}$
  $= 1B5_{16}$

Conversion from Decimal to Binary

- **Use Successive Division**
  - **Repeatedly divide by the desired base until 0 is reached**
  - **Save the remainders as the final answer**
  - **The first remainder is the LSB (least significant bit); the last remainder is the MSB (Most significant bit)**
- $437_{10} = ?_2$

437 / 2 = 218 remainder 1
218 / 2 = 109 remainder 0
109 / 2 = 54 remainder 1
54 / 2 = 27 remainder 0
27 / 2 = 13 remainder 1
13 / 2 = 6 remainder 1
6 / 2 = 3 remainder 0
3 / 2 = 1 remainder 1
1 / 2 = 0 remainder 1

$= 110110101_2$

Conversion from Decimal to Hexadecimal

- $437_{10} = ?_{16}$

437 / 16 = 27 remainder 5

27 / 16 = 1 remainder 11 (11=B)

1 / 16 = 0 remainder 1

- $427_{10} = 1B5_{16}$

Conversion from Binary to Hex

- Starting at the LSB working left, group the bits by 4s. Padding of 0s can occur on the most significant group.
- Convert each group of 4 into the equivalent HEX value.
- %1101110101100 = $?
  = $1BAC

Conversion from Hex to Binary

- Convert each HEX digit to the equivalent 4-bit binary grouping.
- \$A73 = %?

  = %101001110011

Conversion of Fractions

- Conversion from decimal to binary requires multiplying by the desired base (2)
- $0.625_{10} = ?_2$
- $= 0.101_2$

Binary, Octal, and Hexadecimal

- ❖ Binary, Octal, and Hexadecimal are related:

  Radix 16 = 2^4 and Radix 8 = 2^3
- ❖ Hexadecimal digit = 4 bits and Octal digit = 3 bits
- ❖ Starting from least-significant bit, group each 4 bits into a hex digit or each 3 bits into an octal digit
- ❖ Example: Convert 32-bit number into octal and hex

3 5 3 0 5 5 2 3 6 2 4	Octal
1 1 1 0 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 0 0 1 1 1 1 0 0 1 0 1 0 0	32-bit binary
E B 1 6 A 7 9 4	Hexadecimal

Important Properties - cont'd

❖ How many possible values can be represented …

Using n binary digits?	2^n values: 0 to $2^n - 1$
Using n octal digits	8^n values: 0 to $8^n - 1$
Using n decimal digits?	10^n values: 0 to $10^n - 1$
Using n hexadecimal digits	16^n values: 0 to $16^n - 1$
Using n digits in Radix r ?	r^n values: 0 to $r^n - 1$

Addition/Subtraction of Binary Numbers

```
  101011
+      1
  101100
```

```
  101011
+ 001011
  110110
```

- The carry out has a weight equal to the base (in this case 16). The digit that left is the excess (Base – sum).

Addition/Subtraction of Hex Numbers

- $3A
  +$28
  $62

- The carry out has a weight equal to the base (in this case 16). The digit that left is the excess (Base – sum).

Signed Number Representation

- Three ways to represent signed numbers
  - Sign-Magnitude
  - 1s Complement
  - 2s Complement

Sign-Magnitude

- For an N-bit word, the most significant bit is the sign bit; the remaining bits represent the magnitude
- 0110 = +6
- 1110 = -6
- Addition/subtraction of numbers can result in overflow (errors) – (Due to fixed number of bits); two values for zero
- Range for n bits: $-(2^{n-1}–1)$ through $(2^{n-1}–1)$

1s Complement

- Negative numbers = N' = $(2^{n-1}-1)-P$ (where P is the magnitude of the number)
  - For a 5-bit system, -7 = 11111
    -00111
    11000
- Range for n bits:$-(2^{n-1}-1)$ through $(2^{n-1}-1)$

2s Complement

- Negative Numbers = N* = $2^n - P$
  (where P is the magnitude of the number)
  - For a 5-bit system, -7 = 100000
    -00111
    11001
- Another way to form 2s complement representation is to complement P and add 1
- Range for n bits: $-(2^{n-1})$ through $(2^{n-1}-1)$

Numbers Represented with 4-bit Fixed Digit Representation

Decimal	Sign Magnitude	1s Complement	2s Complement
+8			
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	1000	1111	0000
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8			1000

Summary of Signed Number Representations

- Sign Magnitude – has two values for 0
- - errors in addition of negative and positive numbers
- 1s complement – two values for 0
- - additional hardware needed to compensate for this
- 2s Complement – representation of choice

Unsigned/Signed Overflow

- You can detect unsigned overflow if there is a carryout of the MSB.
- You can detect signed overflow if the sum of two positive numbers is a negative number or if the sum of two negative numbers is a positive number. An overflow never occurs in an addition of a negative and a positive number.

Codes

- **Decimal Codes**
  - BCD (Binary Coded Decimal)
    - Weighted Codes (8421, 2421, etc…)
- **ASCII Codes**
  - ASCII (American Standard Code for Information Interchange)
  - Unicode Standard
- **Unit Distance Codes**
  - Gray
- **Error Detection Codes**
  - Parity Bit
- **Error Correction Codes**
  - Hamming Code

BCD Codes (Decimal Codes)

- Coded Representations for the the 10 decimal digits
- Requires 4 bits ($2^3 < 10 < 2^4$)
- Only use 10 combinations of 16 possible combinations

BCD Codes (Decimal Codes)

- **Weighted Code**
  - **8421 code**
    - Most common
    - Default
    - The corresponding decimal digit is determined by adding the weights associated with the 1s in the code group.
    - **The BCD representation is NOT the binary equivalent of the decimal number**
      - **623_{10} = 0110 0010 0011**
  - **2421, 5421,7536, etc… codes**
    - The weights associated with the bits in each code group are given by the name of the code

BCD Codes (Decimal Codes)

- **Nonweighted Codes**
  - **2-out-of-5**
    - Actually weighted 74210 except for the digit 0
    - Used by the post office for scanning bar codes for zip codes
    - Has error detection properties

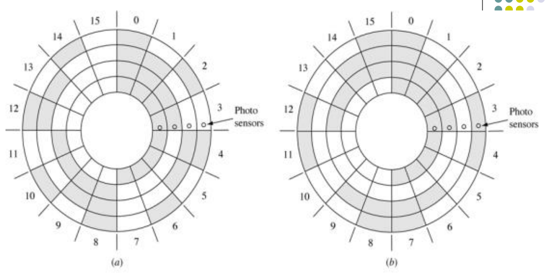
BCD Codes (Decimal Codes)

U.S. Postal Service bar code corresponding to the ZIP code 14263-1045.

Unit Distance Codes

- Important when converting analog to digital
- Only one bit changes between successive integers
- **Gray Code** is most popular example

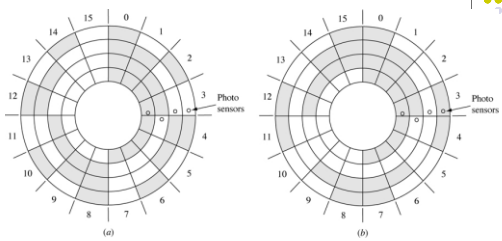
Unit Distance Codes



Angular position encoders.

a) Conventional binary encoder. (*b*) Gray code encoder.

Unit Distance Codes



Angular position encoders with misaligned photosensing devices. (*a*) Conventional binary encoder. (*b*) Gray code encoder.

Mirror Image Conversion

Start	→ Mirror	→ Prefix	→ Mirror	→ Prefix	→ Mirror	→ Prefix
0	0	0 0	0 0	0 0 0	0 0 0	0 0 0 0
1	1	0 1	0 1	0 0 1	0 0 1	0 0 0 1
	1	1 1	1 1	0 1 1	0 1 1	0 0 1 1
	0	1 0	1 0	0 1 0	0 1 0	0 0 1 0
			1 0	1 1 0	1 1 0	0 1 1 0
			1 1	1 1 1	1 1 1	0 1 1 1
			0 1	1 0 1	1 0 1	0 1 0 1
			0 0	1 0 0	1 0 0	0 1 0 0
					1 0 0	1 1 0 0
					1 0 1	1 1 0 1
					1 1 1	1 1 1 1
					1 1 0	1 1 1 0
					0 1 0	1 0 1 0
					0 1 1	1 0 1 1
					0 0 1	1 0 0 1
					0 0 0	1 0 0 0
1-bit		2-bit		3-bit		4-bit

From Binary to a Gray-code

1. Copy MSB
2. Add this bit to the next position
   1. Record Sum
   2. Ignore Carry (if any)
3. Record successive sum until completed

From Binary to Gray code

1 0 1 1

↓ ↓ ↓ ↓

1 1 1 0

From a Gray-code to Binary

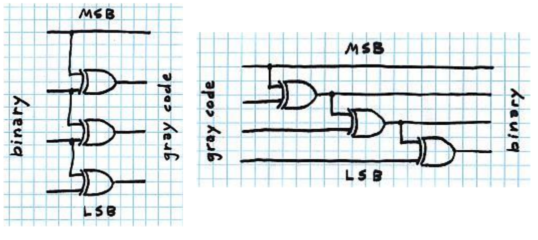
1. Copy MSB
2. Add the Binary MSB to next Significant bit of Gray code
   1. Record Sum
   2. Ignore Carry (if any)
3. Continue process until LSB is reached

From a Gray-code to Binary

1 0 1 1

1 1 0 1

Digital Circuit



Excess-3 code

Decim	Binar	Decim	Binar	Decim	Binar	Decim	Binar
−3	0000	**1**	0100	**5**	1000	**9**	1100
−2	0001	**2**	0101	**6**	1001	**10**	1101
−1	0010	**3**	0110	**7**	1010	**11**	1110
0	0011	**4**	0111	**8**	1011	**12**	1111

Within the range when 1's complemented, apart from BCD

Character Codes

- Character sets
  - Standard ASCII: 7-bit character codes (0 – 127)
  - Extended ASCII: 8-bit character codes (0 – 255)
  - Unicode: 16-bit character codes (0 – 65,535)
  - Unicode standard represents a universal character set
    - Defines codes for characters used in all major languages
    - Used in Windows-XP: each character is encoded as 16 bits
  - UTF-8: variable-length encoding used in HTML
    - Encodes all Unicode characters
    - Uses 1 byte for ASCII, but multiple bytes for other characters
- Null-terminated String
  - Array of characters followed by a NULL character

ASCII Codes

Printable ASCII Codes

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	space	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

- Examples:
  - ASCII code for space character = 20 (hex) = 32 (decimal)
  - ASCII code for 'L' = 4C (hex) = 76 (decimal)
  - ASCII code for 'a' = 61 (hex) = 97 (decimal)

Control Characters

- The first 32 characters of ASCII table are used for control
- Control character codes = 00 to 1F (hexadecimal)
  - Not shown in previous slide
- Examples of Control Characters
  - Character 0 is the NULL character ⇒ used to terminate a string
  - Character 9 is the Horizontal Tab (HT) character
  - Character 0A (hex) = 10 (decimal) is the Line Feed (LF)
  - Character 0D (hex) = 13 (decimal) is the Carriage Return (CR)
  - The LF and CR characters are used together
    - They advance the cursor to the beginning of next line
- One control character appears at end of ASCII table
  - Character 7F (hex) is the Delete (DEL) character

Unicode (Arabic)

Unicode (Bengali)

▯	ঁ	ং	ঃ	▯	অ	আ	ই	ঈ	উ	ঊ	ঋ	ঌ	▯	▯	এ
0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	098A	098B	098C	098D	098E	098F
ঐ	▯	▯	ও	ঔ	ক	খ	গ	ঘ	ঙ	চ	ছ	জ	ঝ	ঞ	ট
0990	0991	0992	0993	0994	0995	0996	0997	0998	0999	099A	099B	099C	099D	099E	099F
ঠ	ড	ঢ	ণ	ত	থ	দ	ধ	ন	▯	প	ফ	ব	ভ	ম	য
09A0	09A1	09A2	09A3	09A4	09A5	09A6	09A7	09A8	09A9	09AA	09AB	09AC	09AD	09AE	09AF
র	▯	ল	▯	▯	▯	শ	ষ	স	হ	▯	▯	়	ঽ	া	ি
09B0	09B1	09B2	09B3	09B4	09B5	09B6	09B7	09B8	09B9	09BA	09BB	09BC	09BD	09BE	09BF
ী	ু	ূ	ৃ	ৄ	▯	▯	ে	ৈ	▯	▯	ো	ৌ	্	▯	▯
09C0	09C1	09C2	09C3	09C4	09C5	09C6	09C7	09C8	09C9	09CA	09CB	09CC	09CD	09CE	09CF
▯	▯	▯	▯	▯	▯	▯	ৗ	▯	▯	▯	▯	ড়	ঢ়	▯	য়

Parity Bit & Error Detection Codes

- ❖ Binary data are typically transmitted between computers
- ❖ Because of noise, a corrupted bit will change value
- ❖ To detect errors, extra bits are added to each data value
- ❖ Parity bit: is used to make the number of 1's odd or even
- ❖ Even parity: number of 1's in the transmitted data is even
- ❖ Odd parity: number of 1's in the transmitted data is odd

7-bit ASCII Character	With Even Parity	With Odd Parity
'A' = 1000001	**0** 1000001	**1** 1000001
'T' = 1010100	**1** 1010100	**0** 1010100

Detecting Errors

Sender	7-bit ASCII character + 1 Parity bit Sent 'A' = 01000001, Received 'A' = 01000101	Receiver

- ❖ Suppose we are transmitting 7-bit ASCII characters
- ❖ A parity bit is added to each character to make it 8 bits
- ❖ Parity can detect all single-bit errors
  - ✧ If even parity is used and a single bit changes, it will change the parity to odd, which will be detected at the receiver end
  - ✧ The receiver end can detect the error, but cannot correct it because it does not know which bit is erroneous
- ❖ Can also detect some multiple-bit errors
  - ✧ Error in an odd number of bits

Hamming Code

1. Number the bits starting from 1: bit 1, 2, 3, 4, 5, etc.
2. Write the bit numbers in binary. 1, 10, 11, 100, 101, etc.
3. All bit positions that are powers of two (have only one 1 bit in the binary form of their position) are parity bits.
4. All other bit positions, with two or more 1 bits in the binary form of their position, are data bits.
5. Each data bit is included in a unique set of 2 or more parity bits, as determined by the binary form of its bit position.

- In general each parity bit covers all bits where the binary AND of the parity position and the bit position is non-zero.

Hamming Code

1. **Parity bit 1** covers all bit positions which have the **least significant bit** set: bit 1 (the parity bit itself), 3, 5, 7, 9, etc.
2. **Parity bit 2** covers all bit positions which have the **second least significant bit** set: bit 2 (the parity bit itself), 3, 6, 7, 10, 11, etc.
3. **Parity bit 4** covers all bit positions which have the **third least significant bit** set: bits 4–7, 12–15, 20–23, etc.
4. **Parity bit 8** covers all bit positions which have the **fourth least significant bit** set: bits 8–15, 24–31, 40–47, etc.

Example

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15
Parity bit coverage	p1	X		X		X		X		X		X		X		X		X		X	
	p2		X	X			X	X			X	X			X	X			X	X	
	p4				X	X	X	X					X	X	X	X					X
	p8								X	X	X	X	X	X	X	X					
	p16																X	X	X	X	X

...

Example

	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7
Data word (without parity):			**0**		**1**	**1**	**0**		**1**	**0**	**1**
p_1	**1**		0		1		0		1		1
p_2		**0**	0			1	0			0	1
p_3				**0**	1	1	0				
p_4								**0**	1	0	1
Data word (with parity):	**1**	**0**	0	**0**	1	1	0	**0**	1	0	1

Calculation of Hamming code parity bits

Example

- The new data word (with parity bits) is now "**10001100101".**
- We now assume the **final bit gets corrupted** and turned from **1 to 0.**
- Our new data word is "**10001100100";**
- How the Hamming codes were created we flag each parity bit as 1 when the even parity check fails.

Example

	p_1	p_2	d_1	p_3	d_2	d_3	d_4	p_4	d_5	d_6	d_7	Parity check	Parity bit
Received data word:	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**		
p_1	**1**		0		1		0		1		0	**Fail**	**1**
p_2		**0**	0			1	0			0	0	**Fail**	**1**
p_3				**0**	1	1	0					Pass	0
p_4								**0**	1	0	0	**Fail**	**1**

Checking of parity bits (switched bit highlighted)

Example

➢The final step is to evaluate the value of the parity bits

➢It goes furthest to the right

➢The integer value of the parity **bits is 11,** signifying that the **11th bit** in the data word (including parity bits) is **wrong** and needs to be **flipped.**

Example

	p_4	p_3	p_2	p_1	
Binary	1	0	1	1	
Decimal	**8**		**2**	**1**	**Σ = 11**

Flipping the 11th bit changes 1000110010**0** back into 1000110010**1**.
Removing the Hamming codes gives the original data word of 0110101.

Warning: Conversion or Coding?

❖ Do NOT mix up conversion of a decimal number to a binary number with coding a decimal number with a binary code

❖ $13_{10} = (1101)_2$ This is conversion

❖ $13 \Leftrightarrow (0001\ 0011)_{BCD}$ This is coding

❖ In general, coding requires more bits than conversion

❖ A number with *n* decimal digits is coded with 4*n* bits in BCD

How Much Memory?

- Memory is purchased in bits –
  - How many bits do I need if I want to distinguish between 8 colors?
  - How many bits do I need if I want to represent 16 million different colors?

How Much Memory?

- How many bits do I need if I want to distinguish between 8 colors?

  $2^{x-1} < 8 <= 2^x$

  x = 3 (3 bits are needed)

- How many bits do I need if I want to represent 16 million different colors?

  $2^{x-1} < 16$ million $<= 2^x$

  $16M = 1Mx16 = 2^{20}x2^4 = 2^{24}$

  x = 24 (24 bits are needed)