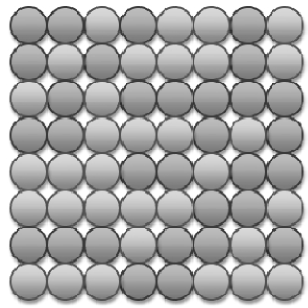


UNIT 7A

Data Representation: Numbers and Text

Digital Data

100101010111110101010110101001110

- What does this binary sequence represent?
- It could be:
  - an integer
  - a floating point number
  - text encoded with ASCII or another standard
  - a pixel of an image
  - several digital samples of a music recording
  - an instruction that the computer is executing
  - ...

Integer Representation

- An integer can be represented using binary.
- An integer can be:
  - unsigned (always considered non-negative)
  - signed (positive or negative)
- An integer can be represented using varying numbers of bits
  - 8 bits (byte)
  - 16 bits (word)
  - 32 bits
  - 64 bits ....

Unsigned Integers

- Every bit represents a power of 2.
- Example (8 bits):

____	____	____	____	____	____	____	____
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	0	1	1	0	1	0	1
2^7		2^5	2^4		2^2		2^0
128 +		32 +	16 +		4 +		1 = 181

Unsigned Integers: Range

bits	minimum	maximum
8	0	$2^8 - 1$ (255)
16	0	$2^{16} - 1$ (65,535)
32	0	$2^{32} - 1$ (4,294,967,295)

Signed Integers

- Every bit represents a power of 2 except the "left-most" bit, which represents the sign of the number (0 = positive, 1 = negative)
- Example for positive integer (8 bits):

0	___	___	___	___	___	___	___
+	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	1	1	0	1	0	0
+		2^5	2^4		2^2		
		32 +	16 +		4		= +52

Signed Integers: 2's complement

- When the leftmost bit is a 1, the integer is negative.
- To find its magnitude, we take the 2's complement of this number.
  – The 2's complement is obtained by flipping each bit of the number (from 0 to 1, or 1 to 0) and then adding 1 to that number.

Signed Integers: Negative

- Example for negative integer (8 bits):

```
 1    1    0    0    1    1    0    0
---
 –   (leftmost bit 1 -> negative)
```

Flip each bit:

```
 0    0    1    1    0    0    1    1
```

and add 00000001 to get magnitude:

```
 0    0    1    1    0    1    0    0
          2^5  2^4       2^2
         32 + 16 +       4            =  52
```

So, **11001100 = –52**

2's complement property

- When you add a number to its 2's complement (in binary), you always get 0.
  - Remember, you're using base 2 arithmetic.
- Example (using 8 bits):

```
    00110100      +52
+   11001100      –52
-------------------------
    00000000        0
```


Signed Integers: Range

bits	minimum	maximum
8	-2^7	$2^7 - 1$
	(–128)	(+127)
	10000000 (binary)	01111111 (binary)
16	-2^{15}	$2^{15} - 1$
	(–32,768)	(+32,767)
32	-2^{31}	$2^{31} - 1$
	(–2,147,483,648)	(+2,147,483,647)

Floating Point Numbers

Age of the Universe in years:

+ $\underline{13.7}$ X 10^{9}

sign — significand — exponent

- Floating point numbers are commonly represented as a binary number with these three components.

IEEE-754 standard

- Most common encoding of floating point numbers on computers today.
- 32-bit ("single-precision") floating point:

```
_  ___________  ______________________________
   exponent          significand
   (8 bits)           (23 bits)
↑
sign
(1 bit)
```

IEEE-754 standard

- Binary Significand
  - Always assumes the form 1.XXXXXXXXX in binary. Does not store the leading 1.
  - Stores the fractional part using 23 bits.
- Exponent
  - Stores exponent offset by 127.
    - Example: An exponent of -6 would be stored as 121.
  - Stores exponent as unsigned 8-bit integer.
  - Exponent range: min -126, max +127

Example: IEEE-754

- Floating point number in binary:
  - 1.<u>0110111</u> X 2^{26}

1 <u>10011001</u> <u>0110111</u>0000000000000000

exponent (8 bits) | **significand (23 bits)**

sign (1 bit)

26 + 127 = 153
binary: 10011001

Text: ASCII standard

- ASCII (American Standard Code for Information Interchange)
  - 7-bit code to represent standard U.S. characters on a keyboard
  - Typically stored using 8 bits.
  - The 8th bit is sometimes used for parity (more on this shortly).

ASCII table

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EN	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
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

- Values above are represented in hexadecimal (base 16).
- ASCII code for "M" is 4D (hex).

ASCII Example

- The ASCII code for "M" is 4D hexadecimal.
- Conversion from base 16 to base 2:

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

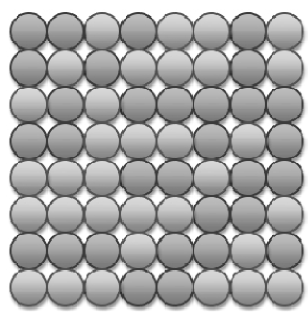
- 4D (hex) = <u>0100</u> <u>1101</u> (binary) = 77 (decimal)
  (leftmost bit can be used for parity)

Parity

- To detect transmission errors, the 8th (leftmost) bit could be used as an error-detection bit.
- Even parity: Set the leftmost bit so that the number of 1's in the byte is even.
- Odd parity: Set the leftmost bit so that the number of 1's in the byte is odd.

Example

- The character "M" is transmitted using odd parity.
- "M" in ASCII (7-bits) is 1001101.
- Using odd parity, we transmit 11001101 since this makes the number of 1's odd.
- If the receiver receives a character with an even number of 1's, the receiver knows something went wrong and requests a retransmission.
  - If two bits are flipped during transmission, we can't detect this with this simple parity scheme, however the probability of 2 or more bits in error is extremely low.



- Seven characters are transmitted here as bytes using even parity along with a special 8th byte.
- The two colors represent 1's and 0's.
- One bit is in error. Can you find it?