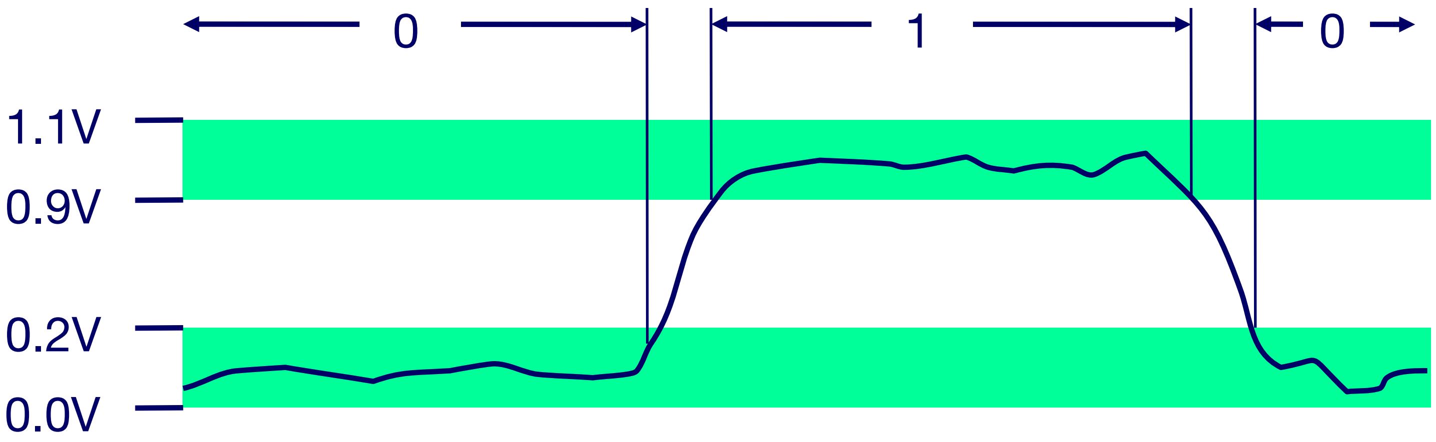
Bits, Bytes, and Integers – Ch2

- **Representing information as bits**
- **Bit-level manipulations**
- **Integers**
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - Addition, negation, multiplication, shifting
  - Summary
- **Representations in memory, pointers, strings**

Everything is bits

- **Each bit is 0 or 1**
- **Why bits? Electronic Implementation**
  - Easy to store with bistable elements
  - Reliably transmitted on noisy and inaccurate wires

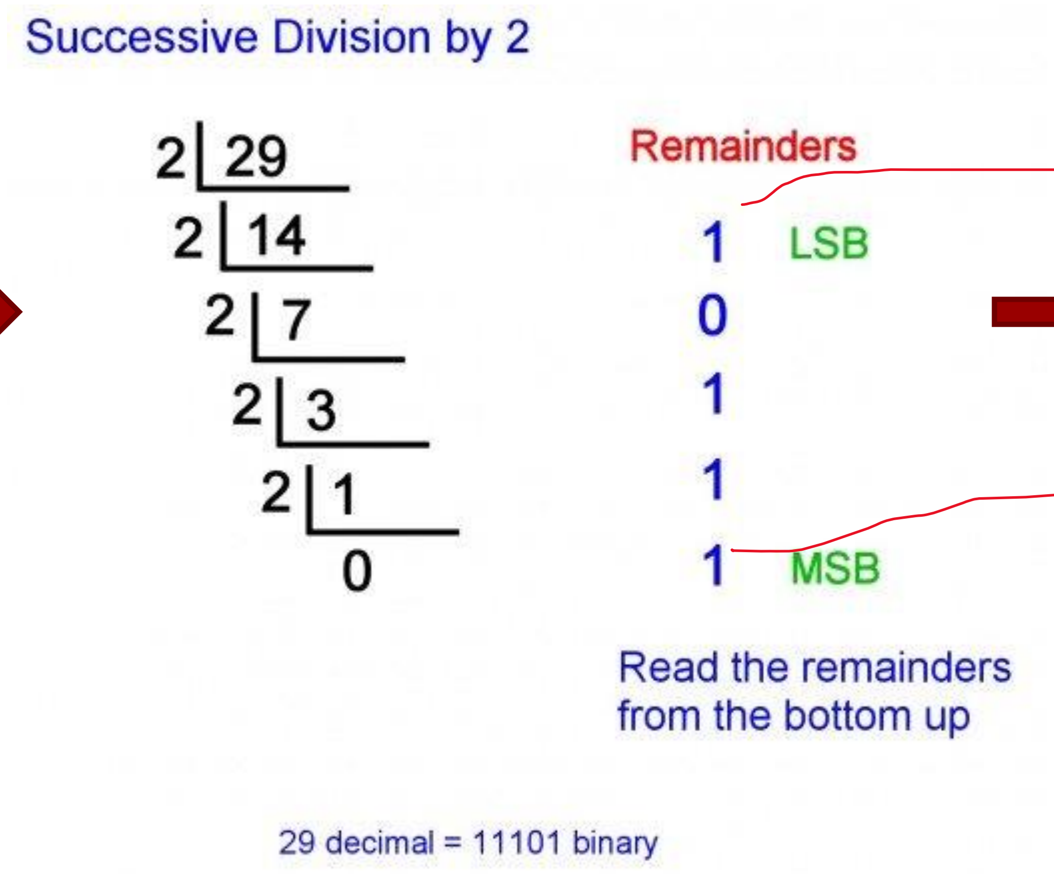


Representing decimal numbers in binary number system

- **Base 2 Number Representation**
  - Represent 15213_{10} as 11101101101101_2
  - Represent 1.20_{10} as $1.0011001100110011[0011]..._2$
  - Represent $1.5213 \text{ X } 10^4$ as $1.1101101101101_2 \text{ X } 2^{13}$

Decimal to Binary conversion by hand

29_{10}



11101_2

Converting decimal to binary Source: © Eugene Brennan

Decimal to Binary conversion for fractions by hand

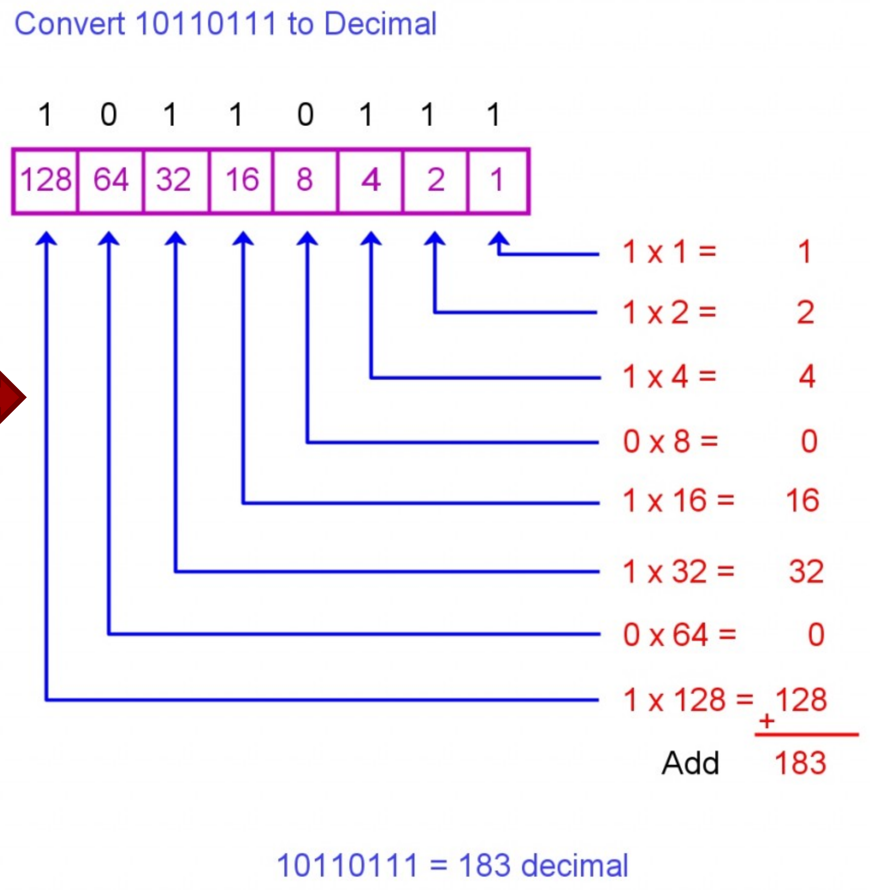
- 0.35_{10}

0.35 X 2	= 0.70	with a carry of	0	Binary Point
0.70 X 2	= 0.40	with a carry of	1	
0.40 X 2	= 0.80	with a carry of	0	Read Down
0.80 X 2	= 0.60	with a carry of	1	
0.60 X 2	= 0.20	with a carry of	1	

Circuit Globe

- $0.010110101101011....{}_{10}$
- $0.\overline{01011}_{10}$

Binary to Decimal conversion by hand

10110111_2 → 

→ 183_{10}

*Converting decimal to binary*Source: © Eugene Brennan

Encoding Byte Values

- Byte = 8 bits
  - **Binary** 00000000_2 to 11111111_2
  - **Decimal**: 0_{10} to 255_{10}
  - **Hexadecimal** 00_{16} to FF_{16}
    - Base 16 number representation
    - Use characters ‘0’ to ‘9’ and ‘A’ to ‘F’
    - Write $FA1D37B_{16}$ in C as
      - 0xFA1D37B
      - 0xfa1d37b

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

Example use in C: https://onlinegdb.com/OIdKx_gJ7

Practice

- **Perform number conversions:**
  - Binary 1001101110011110110101 to hexadecimal
  - 0xD5E4C to binary

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

Practice

- **Perform number conversions:**
  - Binary 10011011100111110110101 to hexadecimal
  - 0xD5E4C to binary

Binary 10011011100111110110101 to hexadecimal:

Binary	10	0110	1110	0111	1011	0101
Hexadecimal	2	6	E	7	B	5

Hexadecimal	D	5	E	4	C
Binary	1101	0101	1110	0100	1100

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

Practice

- Perform the following addition and subtractions in Hex
  - 0x503c + 0x8 =
  - 0x503c – 0x40 =

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

Practice

- **Perform the following addition and subtractions in Hex**
  - 0x503c + 0x8 =
  - 0x503c – 0x40 =

  - 0x503c + 0x8 = 0x5044. Adding 8 to hex c gives 4 with a carry of 1.
  - 0x503c – 0x40 = 0x4ffc. Subtracting 4 from 3 in the second digit position requires a borrow from the third. Since this digit is 0, we must also borrow from the fourth position.

Today: Bits, Bytes, and Integers

- Representing information as bits
- **Bit-level manipulations**
- Integers
  - Representation: unsigned and signed
  - Conversion, casting
  - Expanding, truncating
  - Addition, negation, multiplication, shifting
  - Summary
- Representations in memory, pointers, strings

Boolean Algebra

- **Developed by George Boole in 19th Century**
  - Algebraic representation of logic
    - Encode "True" as 1 and "False" as 0

And

- **A&B = 1 when both A=1 and B=1**

&	0	1
0	0	0
1	0	1

Or

- **A|B = 1 when either A=1 or B=1**

|	0	1
0	0	1
1	1	1

Not

- **~A = 1 when A=0**

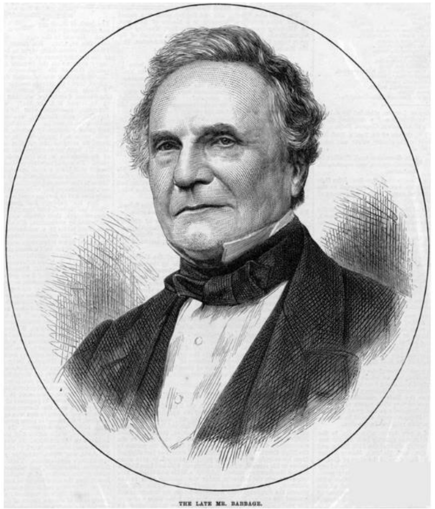
~	
0	1
1	0

Exclusive-Or (Xor)

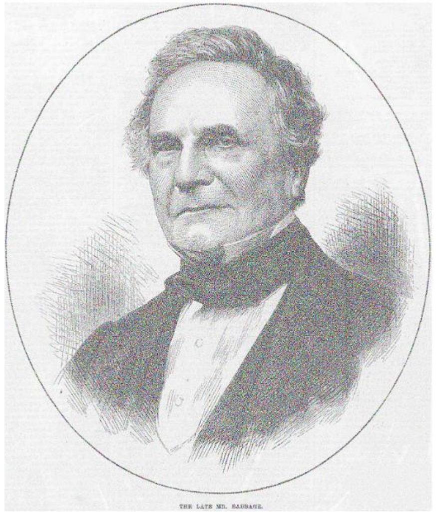
- **A^B = 1 when either A=1 or B=1, but not both**

^	0	1
0	0	1
1	1	0

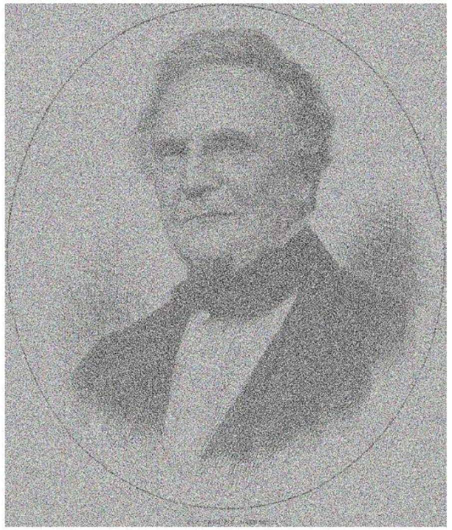
Guess the operation



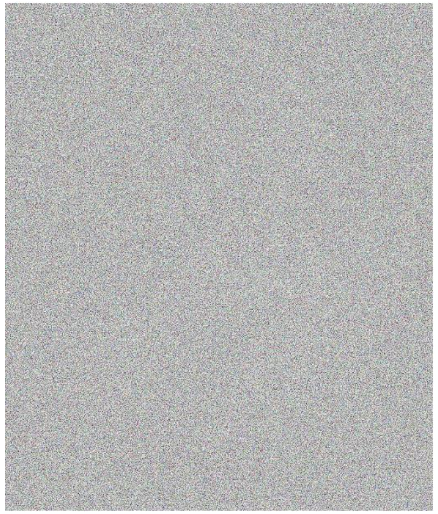
Original



Original OR random stream



Original AND Random stream



Original XOR random stream

Resource: khanacademy.org

Bitwise Operations

- **Operate on Bit Vectors**
  - Operations applied bitwise

```
  01101001     01101001     01101001
& 01010101   | 01010101   ^ 01010101   ~ 01010101
  01000001     01111101     00111100     10101010
```

- **All of the Properties of Boolean Algebra Apply**

Boolean Laws

T1 : Commutative Law

(a) $A + B = B + A$

(b) $A \& B = B \& A$

T2 : Associative Law

(a) $(A + B) + C = A + (B + C)$

(b) $(A \& B) \& C = A \& (B \& C)$

T3 : Distributive Law

(a) $A \& (B + C) = A \& B + A \& C$

(b) $A + (B \& C) = (A + B) \& (A + C)$

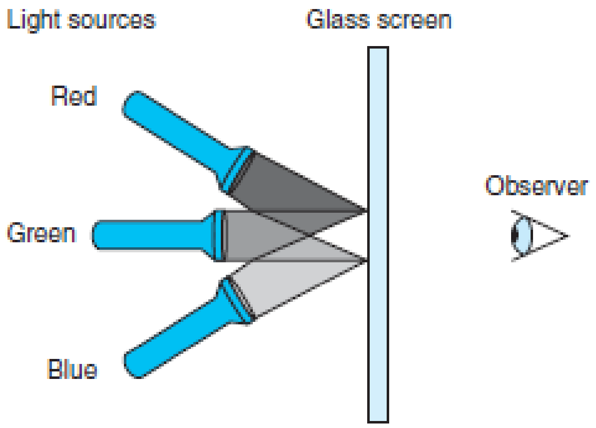
T4 : Identity Law

(a) $A + A = A$

(b) $A \& A = A$

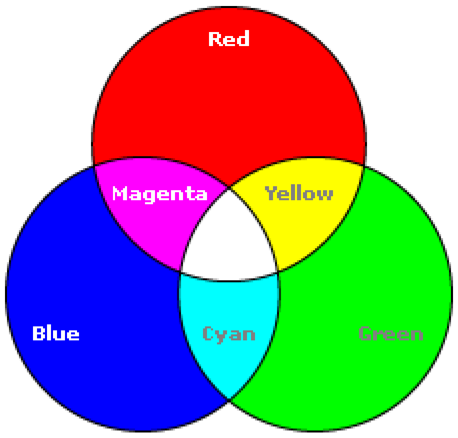
Practice

- **We can create eight different colors based on the absence (0) or presence (1) of light sources *R*, *G*, and *B*:**



- **Describe the effect of applying Boolean operations on the following colors:**
  - Blue | Green =
    - Cyan (011)
  - Yellow & Cyan =
    - Green (010)
  - Red ^ Magenta =
    - Blue (001)

RGB color model



R	*G*	*B*	Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

Bit-Level Operations in C

- **Operations &, |, ~, ^ Available in C**
  - Apply to any “integral” data type
    - long, int, short, char, unsigned
  - View arguments as bit vectors
  - Arguments applied bit-wise
- **Examples (Char data type)**
  - ~0x41 -> 0xBE
    - $\sim 01000001_2 \rightarrow 10111110_2$
  - ~0x00 -> 0xFF
    - $\sim 00000000_2 \rightarrow 11111111_2$
  - 0x69 & 0x55 -> 0x41
    - 01101001_2 & $01010101_2 \rightarrow 01000001_2$
  - 0x69 | 0x55 -> 0x7D
    - $01101001_2 \mid 01010101_2 \rightarrow 01111101_2$

```c
#include <stdio.h>


void main()
{
    unsigned char A = 'A';
    unsigned char Anot = ~A;
    printf("0x%x\n",A);
    printf("0x%x\n",Anot);

}
```

x represents unsigned hex integer

```
0x41
0xbe
```

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

Contrast: Logic Operations in C

Contrast to Logical Operators

- &&, ||, !
  - View 0 as “False”
  - Anything nonzero as “True”
  - Always return 0 or 1
  - Early termination

Examples (char data type)

- !0x41 -> 0x00
- !0x00 -> 0x01
- !!0x41 -> 0x01

- 0x69 && 0x55 -> 0x01
- 0x69 || 0x55 -> 0x01

Contrast: Logic Operations in C

Contrast to Logical Operators

- &&, ||, !
  - View 0 as “False”
  - Anything nonzero as “True”
  - Always return 0 or 1
  - Early termination

Examples (char data

- !0x41 -> 0x00
- !0x00 -> 0x01
- !!0x41 -> 0x01

- 0x69 && 0x55 -> 0x01
- 0x69 || 0x55 -> 0x01

Watch out for && vs. & (and || vs. |)…
one of the common oopsies in
C programming

Practice 2.14

- Suppose that x and y have byte values 0x66 and 0x39, respectively. Fill in the following table indicating the byte values of the different C expressions:

Expression	Value	Expression	Value
`x & y`	______	`x && y`	______
`x | y`	______	`x || y`	______
`~x | ~y`	______	`!x || !y`	______
`x & !y`	______	`x && ~y`	______

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

Practice 2.14

- Suppose that x and y have byte values 0x66 and 0x39, respectively. Fill in the following table indicating the byte values of the different C expressions:

Expression	Value	Expression	Value
x & y	0x20	x && y	0x01
x | y	0x7F	x || y	0x01
~x | ~y	0xDF	!x || !y	0x00
x & !y	0x00	x && ~y	0x01

```
X = 0110 0110        ~X = 1001 1001
Y = 0011 1001        ~Y = 1100 0110

  0110 0110            1001 1001
& 0011 1001          | 1100 0110


  0110 0110
| 0011 1001
```

Shift Operations

- **Left Shift: x << y**
  - Shift bit-vector **x** left **y** positions
      - Throw away extra bits on the left
    - Fill with 0's on the right
- **Right Shift: x >> y**
  - Shift bit-vector **x** right **y** positions
    - Throw away extra bits on the right
  - Logical shift
    - Fill with 0's on the left
  - Arithmetic shift
    - Replicate most significant bit on the left
- **Undefined Behavior**
  - Shift amount < 0 or ≥ word size

Argument **x**	**01100010**
<< 3	**00010*000***
Log. **>> 2**	***00*011000**
Arith. **>> 2**	***00*011000**

Argument **x**	**10100010**
<< 3	**00010*000***
Log. **>> 2**	***00*101000**
Arith. **>> 2**	***11*101000**

In C:
>> unsigned is logical
>> signed is Arithmetic

Shift Operations

```c
unsigned int num1 = 64;
int num2 = -64;

printf("0x%08x  %d\n", num1, num1);
printf("0x%08x  %d\n", num1>>4, num1>>4);
printf("0x%08x  %d\n", num1<<4, num1<<4);
printf("\n");
printf("0x%x  %d\n", num2, num2);
printf("0x%x  %d\n", num2>>4, num2>>4);
printf("0x%x  %d\n", num2<<4, num2<<4);
```

```
0x00000040   64
0x00000004   4
0x00000400   1024

0xffffffc0   -64
0xfffffffc   -4
0xfffffc00   -1024
```

- Left shift is always equivalent to multiplication by 2
- Right shift is always equivalent to division by 2 **for even numbers**.

Today: Bits, Bytes, and Integers

- Representing information as bits
- Bit-level manipulations
- **Integers**
  - **Representation: unsigned and signed**
  - Conversion, casting
  - Expanding, truncating
  - Addition, negation, multiplication, shifting
  - Summary
- Representations in memory, pointers, strings
- Summary