The University Of Alabama in Huntsville Computer Science

cs309

# Chapter 1

G. W. Cox – Spring 2010

The University Of Alabama in Huntsville Computer Science

# Digital Logic

cs309

- Electronic devices used to build computer circuits
  - Can take on Two States represented by voltage levels (high and low).
  - We can think of these as representing any 2-valued system:
    - “1” or “0” (“binary digits” or bits)
    - “True” or “False”
    - “on” or “off”
    - “live” or “dead”
  - We can build on these 2-states to implement:
    - Number representations and arithmetic operations (binary numbers)
    - Codes for various purposes (character codes, error codes, addresses…)
    - Logical operations

G. W. Cox – Spring 2010

# Grouping bits



The University Of Alabama in Huntsville Computer Science

- We can group bits to represent larger numbers of things:

| | | | | | |
|---|---|---|---|---|---|
| 0 | Black | 00 | Black | 000 | Black |
| 1 | White | 01 | White | 001 | White |
| | | 10 | Blue | 010 | Blue |
| | | 11 | Red | 011 | Red |
| | | | | 100 | Yellow |
| | | | | 101 | Orange |
| | | | | 110 | Pink |
| | | | | 111 | Green |

*With n bits, we can represent $2^n$ values or give a unique identifier to $2^n$ things*

- Common terms of reference:

bit: 1 binary digit
byte: 8 bits
word: often 16 bits (may vary)



# Counting in Binary



The University Of Alabama in Huntsville Computer Science

| Decimal | Binary |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 10 |
| 3 | 11 |
| 4 | 100 |
| 5 | 101 |
| 6 | 110 |
| 7 | 111 |
| 8 | 1000 |
| 9 | 1001 |
| 10 | 1010 |
| 11 | 1011 |
| 12 | 1100 |
| 13 | 1101 |
| 14 | 1110 |
| 15 | 1111 |

*Note that with n bits,we can give a unique number to $2^n$ things, but we can only count to $2^n$ - 1*

# Binary Numbers

*The University Of Alabama in Huntsville Computer Science*

**cs309**

To represent numbers, we use Binary (Base 2) numbers

Written using positional notation, similar to decimal numbers:

**Base 10 (Decimal)**

$7309.5_{10} = 7x10^{3} + 3x10^{2} + 0x10^{1} + 9x10^{0} + 5x10^{-1}$

**Base 2 (Binary)**

$1101.101_{2} = 1x2^{3} + 1x2^{2} + 0x2^{1} + 1x2^{0} + 1x2^{-1} + 0x2^{-2} + 1x2^{-3}$

**Base r**

$abc.de = axr^{2} + bxr^{1} + cxr^{0} + dxr^{-1} + exr^{-2}$

Note: Another name for "base" is "radix"

*G. W. Cox – Spring 2010*

# Converting from binary to decimal

*The University Of Alabama in Huntsville Computer Science*

**cs309**

**The positional notation gives an easy way to convert from a binary number to the decimal equivalent.**

$1101.101_{2} = 1x2^{3} + 1x2^{2} + 0x2^{1} + 1x2^{0} + 1x2^{-1} + 0x2^{-2} + 1x2^{-3}$

$= 8 + 4 + 0 + 1 + 0.5 + 0 + 0.125$

$= 13.625_{10}$

*G. W. Cox – Spring 2010*

# Powers of 2

The University Of Alabama in Huntsville Computer Science

cs309

Some key values (in decimal):

$2^0$ = 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 = "1 K"

$2^{20}$ = "1 M"

$2^{30}$ = "1 G"

Know these!

Be careful:

| common computer terminology | common scientific terminology |
|---|---|
| "1K" = $2^{10}$ = 1,024 | "1K" = $10^3$ = 1,000 |
| "1M" = $2^{20}$ = 1,048,576 | "1M" = $10^6$ = 1,000,000 |
| "1G" = $2^{30}$ | "1G" = $10^9$ |

G. W. Cox – Spring 2010

The University Of Alabama in Huntsville Computer Science

cs309

# Binary Arithmetic

G. W. Cox – Spring 2010

*The University Of Alabama in Huntsville Computer Science*

# Binary addition

**cs309**

Addition works exactly the same way as in decimal

```
c2   c1
     a1   a0
     b1   b0
-----------
s2   s1   s0
```

Example:

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

**check:**

$101101 = 2^5+2^3+2^2+2^0 = 32+8+4+1 = 45_{10}$

$100111 = 2^5+2^2+2^1+2^0 = 32+4+2+1 = 39_{10}$

-------                                   ---

$1010100 = 2^6+2^4+2^2 = 64+16+4 = 84_{10}$

*G. W. Cox – Spring 2010*

*The University Of Alabama in Huntsville Computer Science*

# Binary subtraction

**cs309**

Subtraction also works the same way as in decimal

```
                 10
           0    ~0~   10
  1   0   ~1~   ~1~   ~0~   1
- 1   0    0     1     1    1
------------------------------
  0   0    0     1     1    0
```

**check:**

$101101 = 45_{10}$

$100111 = 39_{10}$

--------   ---

$110 = 6_{10}$

*G. W. Cox – Spring 2010*

# Binary multiplication

The University Of Alabama in Huntsville Computer Science

cs309

Same method, but much simpler than decimal multiplication because you only ever multiply by “1” to get the interim results

Starting at the right end of the multiplier (bottom number), examine each bit.
- If 0, skip
- if 1, copy the multiplicand right-aligned under the multipier bit you are examining

After all multiplier bits examined, add the interim results

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



# Binary Division

The University Of Alabama in Huntsville Computer Science

cs309

Just like decimal division

```
               0 1 1 1
          ______________
1 1 1   |  1 1 0 1 0 1
        | - 0 0 0
           1 1 0 1
        -    1 1 1
             1 1 0 0
           - 1 1 1
               1 0 1 1
             - 1 1 1
                 1 0 0 (remainder)
```

G. W. Cox – Spring 2010

# A shortcut: Multiplying/Dividing by the radix

cs309

The University Of Alabama in Huntsville Computer Science

For a number in base "r",
  shifting left 1 place multiplies by r
  shift right one place divides by r

Decimal:
  503 shifted left = 5030.0
  503 shifted right = 50.3
  500 shifted right three times = 0.503 (= 503 / $10^3$)

Binary:
  1101 shifted left = 11010 ($13_{10}$ x 2 = $26_{10}$)
  1101 shifted right = 110.1 ($13_{10}$ / 2 = $6.5_{10}$)

G. W. Cox – Spring 2010

cs309

The University Of Alabama in Huntsville Computer Science

# Octal and Hexadecimal

G. W. Cox – Spring 2010

The University Of Alabama in Huntsville Computer Science

# Base 8 (Octal)

cs309

| Octal | Decimal Equivalent |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 10 | 8 |
| 11 | 9 |
| 12 | 10 |
| 13 | 11 |
| 14 | 12 |
| 15 | 13 |
| 16 | 14 |
| 17 | 15 |
| 20 | 16 |

Example:

$147_8 = 1 \times 8^2 + 4 \times 8^1 + 7 \times 8^0$
$= 64 + 32 + 7$
$= 103_{10}$

*G. W. Cox – Spring 2010*

The University Of Alabama in Huntsville Computer Science

# Base 16 (Hexadecimal or “Hex”)

cs309

| Hex Digit | Decimal Equivalent |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| A | 10 |
| B | 11 |
| C | 12 |
| D | 13 |
| E | 14 |
| F | 15 |

Examples:

$21_{16} = 2 \times 16^1 + 1 \times 16^0 = 33_{10}$

$5F_{16} = 5 \times 16^1 + 15 \times 16^0 = 95_{10}$

$B65F_{16} = 11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0$
$= 45{,}056 + 1{,}536 + 80 + 15$
$= 46{,}687_{10}$

*G. W. Cox – Spring 2010*

The University Of Alabama in Huntsville Computer Science

cs309

# Converting between bases

G. W. Cox – Spring 2010

The University Of Alabama in Huntsville Computer Science

# Converting from base "r" to Decimal

cs309

- Use the power series expansion:

$$N = (d_2\ d_1\ d_0 \mathbf{.}\ d_{-1}\ d_{-2})_r$$
$$= (d_2 \times r^2 + d_1 \times r^1 + d_0 \times r^0 \mathbf{.}\ d_{-1} \times r^{-1} + d_{-2} \times r^{-2})_{10}$$

G. W. Cox – Spring 2010

The University Of Alabama in Huntsville Computer Science

# Converting Integers from Decimal to Base "r"

cs309

For integers, use the "radix divide" technique:

To convert a decimal integer N to base "r",

1. Divide N by r, recording the remainder.
2. If the quotient from the previous step >0, divide it by r, record the remainder and repeat.
3. Write down the remainders from last to first.

$245_{10} = ?_2$

```
2 |245
2 |122   1 ^
 2 | 61  0 |
 2 | 30  1 |
 2 | 15  0 |   = 1 1 1 1  0 1 0 1_2
  2|  7  1 |
  2|  3  1 |
  2|  1  1 |
      0  1 |
```



The University Of Alabama in Huntsville Computer Science

# Converting Fractions from Decimal to Base "r"

cs309

For fractions, use the "radix multiply" technique:

To convert a decimal fraction f to base "r",

1. Multiply f by r. Record the integer part of the result.
2. If the fractional part of the result from the previous step =0 or you have the desired number of digits, stop. Else, multiply the fractional part by r, record the integer part and repeat.
3. Write down the integer digits from first to last.

$0.345_{10} = ?_2$ (5 fractional bits)

```
                        recorded integer part
0.345 x 2 = 0.690   0 |
0.690 x 2 = 1.380   1 |
0.380 x 2 = 0.760   0 |   = 0 . 0 1 0 1 1_2
0.760 x 2 = 1.520   1 |
0.520 x 2 = 1.040   1 v
```

*This is 0.34375, not 0.345. The difference is caused by the roundoff error resulting from only using 5 bits to represent the fraction.*

The University Of Alabama in Huntsville Computer Science

## Converting mixed numbers from decimal to Base "r"

cs309

Use the Radix Divide technique for the integer part, Radix Multiply for the fractional part.

Example:

From previous 2 slides

$245.345_{10} = 11110101.01011_2$

G. W. Cox – Spring 2010

The University Of Alabama in Huntsville Computer Science

## Converting between arbitrary bases

cs309

To determine: $N_p = ?_q$

Use Power Series Expansion to convert N to decimal, then use Radix Multiply/Divide to convert to base q.

$43_5 = ?_3$

$43_5 = 4x5^1 + 3x5^0 = 23_{10}$

```
3 |23
 3 |7   2  ↑
 3 |2   1  |    = 212_3
    0   2  |
```

G. W. Cox – Spring 2010

# A shortcut for Binary, Octal and Hex



A single digit in base $2^p$ expands to its equivalent p bits in binary.
This lets us convert directly between binary, hex, and octal

$553.76_8 = ?_2$

5 5 3 . 7 $6_8$

101 101 011 . 111 $110_2$

$C6B.2_{16} = ?_2$

C 6 B . $2_{16}$

1100 0110 1011 . $0010_2$

$553.76_8 = ?_{16}$

5 5 3 . 7 $6_8$

000101 101 011 . 111 11000

1 6 B . F $8_{16}$

*Count groups outward from the radix point, not from the left and right end*







# Other Notational Systems

## Signed Magnitude

The University Of Alabama in Huntsville Computer Science

cs309

Notation: "$N_{sm}$"

The first bit shows the sign of N, "0" for positive, "1" for negative.

The rest of the bits are |N|.

Example:

$+(1101)_2 = (0\ 1101)_{sm}$

$-(1101)_2 = (1\ 1101)_{sm}$

G. W. Cox – Spring 2010

## 1's Complement

The University Of Alabama in Huntsville Computer Science

cs309

Notation: $N_{1s}$

The first bit shows the sign of N, as in Signed Magnitude.

If the number is positive, the rest of the bits are |N|.

If the number is negative, the rest of the bits are $(2^n - 1) - |N|$ where n is the # of bits in |N|.

Example:

$+(1101)_2 = (0\ 1101)_{1s}$

$-(1101)_2$
we determine $(2^4 – 1) - 1101 = 1111 - 1101 = 0010$
then $-(1101)_2 = (1\ 0010)_{1s}$

*Note that we can obtain $(2^n - 1) - |N|$ by replacing every "1" in |N| with "0" and every "0" with "1". This is called taking the bit-by-bit complement of |N|.*

*General concept: The complement of 0 is 1, the complement of 1 is 0.*

G. W. Cox – Spring 2010

## 2's Complement

cs309

Notation: $N_{2s}$

The first bit shows the sign of N, as in Signed Magnitude.

If the number is positive, the rest of the bits are |N|.

If the number is negative, the rest of the bits are $(2^n - 1) - |N| + 1$ where n is the # bits in |N|.

Examples:

$+(1101)_2 = (0\ 1101)_{2s}$

$-(1101)_2$ → 1111 - 1101 + 1 = 0011 → $-(1101)_2 = (1\ 0011)_{1s}$

*Two shortcuts to determine $(2^n-1) - |N| + 1$:*

*(1) Take the bit-by-bit complement of |N| and add 1*
*(2) Start at the right end of |N|, copy all 0's moving left. Copy the first 1. Complement all other bits.*

The University Of Alabama in Huntsville Computer Science

G. W. Cox – Spring 2010

## Why complement number systems?

cs309

If numbers are represented in 1's or 2's complement number systems, numbers of either sign will add correctly.

The University Of Alabama in Huntsville Computer Science

G. W. Cox – Spring 2010

The University Of Alabama in Huntsville Computer Science

# Arithmetic using 2's complement

cs309

To add two numbers in 2's complement, add them (including the sign bits) as if they are normal binary numbers and ignore the carry, if any. Surprisingly, the sign bit will always be correct, unless we overflow (see below).

$$\begin{array}{rr} 5_{10} & 0\ 0101_{2s} \\ +4_{10} & 0\ 0100_{2s} \\ \hline 9_{10} & 0\ 1001_{2s} \end{array} \qquad \begin{array}{rr} 5_{10} & 0\ 0101_{2s} \\ +\ -4_{10} & 1\ 1100_{2s} \\ \hline 1_{10} & \not{1}0\ 0001_{2s} \end{array}$$

$$\begin{array}{rr} -5_{10} & 1\ 1011_{2s} \\ +\ 4_{10} & 0\ 0100_{2s} \\ \hline -1_{10} & 1\ 1111_{2s} \end{array} \qquad \begin{array}{rr} -5_{10} & 1\ 1011_{2s} \\ +\ -4_{10} & 1\ 1100_{2s} \\ \hline -9_{10} & \not{1}1\ 0111_{2s} \end{array}$$

$$\begin{array}{rr} 15_{10} & 0\ 1111_{2s} \\ +\ 15_{10} & 0\ 1111_{2s} \\ \hline 30_{10} & 1\ 1110_{2s} \end{array}$$

This is actually $-2_{10}$, not $+30_{10}$. The problem is that we cannot represent 30 with only 4 magnitude bits, so the addition "overflowed". <u>When you add two 2's complement numbers of the same sign and you get the opposite sign for the result</u>, you have an overflow condition.



The University Of Alabama in Huntsville Computer Science

# Arithmetic using 1's complement

cs309

To add two numbers in 1's complement, add them (including the sign bits) and if there is a carry, add it into the least-significant bit (the bit at the right end).

$$\begin{array}{rr} 5_{10} & 0\ 0101_{1s} \\ +4_{10} & 0\ 0100_{1s} \\ \hline 9_{10} & 0\ 1001_{1s} \end{array} \qquad \begin{array}{rr} 5_{10} & 0\ 0101_{1s} \\ +\ -4_{10} & 1\ 1011_{1s} \\ \hline 1_{10} & 10\ 0000 \\ & \rightarrow 1 \\ \hline & 0\ 0001_{1s} \end{array}$$

$$\begin{array}{rr} -5_{10} & 1\ 1010_{1s} \\ +\ 4_{10} & 0\ 0100_{1s} \\ \hline -1_{10} & 1\ 1110_{1s} \end{array} \qquad \begin{array}{rr} -5_{10} & 1\ 1010_{1s} \\ +\ -4_{10} & 1\ 1011_{1s} \\ \hline -9_{10} & 11\ 0101 \\ & \rightarrow 1 \\ \hline & 1\ 0110_{1s} \end{array}$$

Overflow is defined the same as in 2's Complement

The University Of Alabama in Huntsville Computer Science

cs309

# Other Binary Codes

G. W. Cox – Spring 2010

The University Of Alabama in Huntsville Computer Science

# Binary Coded Decimal (BCD)

cs309

A way of representing decimal numbers in digital logic

Each digit of the decimal number is represented by 4 bits

| Digit | BCD code |
|---|---|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |
| 8 | 1000 |
| 9 | 1001 |

Example:

$134_{10} = (0001\ 0011\ 0100)_{BCD}$

G. W. Cox – Spring 2010

# BCD Addition

*The University Of Alabama in Huntsville Computer Science*

cs309

To add 2 multiple-digit BCD numbers, first add them as if they were normal binary numbers.
Then check each group of 4 bits. If the group is 9 ('1001') or less, it is OK.
When a group of 4 bits is 10 or more, add 6, adding any carry into the next group of 4 bits.

**Example:**

| Decimal | BCD |
|---|---|
| 184 | 0001 1000 0100 |
| + 576 | 0101 0111 0110 |
| | 0110 1111 1010 |
| | 0110 0110 |
| 760 | 0111 0110 0000 |

These two groups of 4 bits are >9, so we have to adjust by adding 6 to each of them.

*G. W. Cox – Spring 2010*

# Excess-3 code

*The University Of Alabama in Huntsville Computer Science*

cs309

Another way of representing decimal numbers.

Each digit of the decimal number is represented by 4 bits, equal to the BCD code plus 3.

| Digit | Excess-3 code |
|---|---|
| 0 | 0011 |
| 1 | 0100 |
| 2 | 0101 |
| 3 | 0110 |
| 4 | 0111 |
| 5 | 1000 |
| 6 | 1001 |
| 7 | 1010 |
| 8 | 1011 |
| 9 | 1100 |

Example:

$134_{10} = (0100\ 0110\ 0111)_{Excess\text{-}3}$

*G. W. Cox – Spring 2010*

The University Of Alabama in Huntsville Computer Science

# Gray Code

cs309

Designed so that only one bit changes when going between two consecutive numbers

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

*G. W. Cox – Spring 2010*

The University Of Alabama in Huntsville Computer Science

# Alphanumeric codes

cs309

Used to represent alphabetical, number, punctuation, symbols in digital systems (also some common control signals)

Most widely used: American Standard Code for Information Exchange (ASCII)

See text page 24 for list.

Some examples:

| Symbol | ASCII code |
|---|---|
| "A" | 100 0001 |
| "a" | 110 0001 |
| "2" | 011 0010 |
| "!" | 010 0001 |
| "line feed" | 000 1010 |
| "delete" | 111 1111 |

*G. W. Cox – Spring 2010*

## Error-detecting codes

cs309

The University Of Alabama in Huntsville Computer Science

Designed to allow detection of some types of errors in digital data.

Example:

Odd parity: add a bit to the data item to make the total number of 1's odd

ASCII "A" = 100 0001

with odd parity bit: 1100 0001

We will talk about error correction codes later this term.

G. W. Cox – Spring 2010

cs309

The University Of Alabama in Huntsville Computer Science

## Binary Logic

G. W. Cox – Spring 2010

# The Basic Logic Operations

cs309

The University Of Alabama in Huntsville Computer Science

**AND** x·y or xy

| x | y | xy |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

*This is a "Truth Table" (The operation's result for every combination of values)*

**OR** x + y

| x | y | x+y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**NOT** or **COMPLEMENT** or **INVERT** x'

| x | x' |
|---|---|
| 0 | 1 |
| 1 | 0 |

*G. W. Cox – Spring 2010*

# Using Logic Expressions

cs309

The University Of Alabama in Huntsville Computer Science

We can write logical and arithmetic expressions by combining logic values, variables, and operations

Example:
"If it's Winter, the grass is brown. If it's Summer and it's a drought, the grass is brown. Else, the grass is green."

Define variables:

S=0 if it's Winter, S=1 if it's Summer
D=1 if there's a drought, D=0 if there's no drought.
B=1 if the grass is brown, B=0 if the grass is green

Truth Table:

| S | D | B |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

⇨

Logic Expression:

B = S'D' + S'D + SD

*G. W. Cox – Spring 2010*

# Using Logic Expressions (2)

cs309

The University Of Alabama in Huntsville Computer Science

Example (arithmetic):

```
   a
+  b
 c s
```

⇨

Truth Table:

| a | b | c | s |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Logic Expressions:

c = ab

s = a'b + ab'



# Digital Logic Circuits

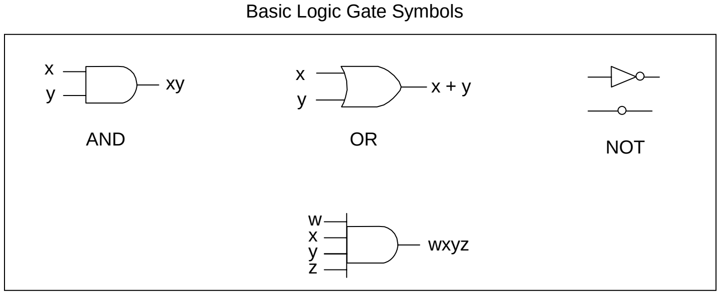
cs309

The University Of Alabama in Huntsville Computer Science

Implementation of Logic variables and operations in electronics

- Values (0 and 1) represented by voltage levels
- Variables implemented by electrical connections
- Operations implemented by logic gates

Basic Logic Gate Symbols




G. W. Cox – Spring 2010