CS 230 – Introduction to Computers and Computer Systems

Module 1 – Arithmetic, Hardware, Data

Overview

- number representation
- boolean algebra and gate logic
- integer arithmetic
- non-numerical data types
- floating point

Number Representation

- *radix representation*
  - radix also know as *base*
- writing natural numbers using a finite alphabet
- given an n-digit word in base *r*

a collection of symbols, e.g. in decimal system, 0-9 is the alphabet and 1 is a symbol

descending, zero-indexed positioning

$$d_{n-1} d_{n-2} d_{n-3} \ldots d_3 d_2 d_1 d_0$$

- integer value is $\sum_{0}^{n-1} d_i * r^i$

value in base 10

Radix Representation

- humans: base-10, decimal
  - why? because humans have 10 fingers/digits on their hands
- computers: base-2, binary
  - why? electrical simplicity allows for logical operations to be performed by electrical switching circuits
    - analog/digital conversion (high vs. low voltage)
  - low-level decimal conversion? Only if necessary
    - storage expansion / waste

Examples

$135_{\mathbf{dec}}$ $$5 \cdot 10^0 + 3 \cdot 10^1 + 1 \cdot 10^2 = 135$$

- not too surprising...

$1440_{\mathbf{sep}}$ $$0 \cdot 7^0 + 4 \cdot 7^1 + 4 \cdot 7^2 + 1 \cdot 7^3 = 567$$

$A32_{\mathbf{hex}}$ $$2 \cdot 16^0 + 3 \cdot 16^1 + 10 \cdot 16^2 = 2610$$

- use letters A...F to express digits > 9 i.e. double-digit numbers
- A-10, B-11, C-12, D-13, E-14, F-15

Conversion from Decimal

- repeatedly divide by target base
- remainders generate digits
  - from right to left...
- example: $3219_{\textbf{dec}}$

$$
\begin{aligned}
3219/16 &= 201 \quad R\,3 \\
201/16 &= 12 \quad R\,9 \\
12/16 &= 0 \quad R\,12
\end{aligned}
$$

$= C93_{\textbf{hex}}$

```
def convert_frm_dec(base, num):
    quo = num
    digits = [ ]
    while quo != 0:
        digits.append(quo mod base)
        quo = quo // base
    final = 0
    for i in range(len(digits)):
         final += digits[i] * (10 ** i) #unless hex
    return final
```

Binary Numbers

- only 0 and 1 as digits – represent low and high voltage
- example

$11101100_{\mathbf{bin}}$

$$2^2 + 2^3 + 2^5 + 2^6 + 2^7 = 236$$

Equivalent to: 0(2^0) + 0(2^1) + 1(2^2) + …
Since 0 and 1, we can skip digits that are 0

- permits simple binary operations

Easy way to convert to dec for binary for small binary values: write the value of 2^i above the i-th digit, and add the products of the digits and values together

Binary / Hex Conversion

$0000_{\textbf{bin}} = 0_{\textbf{hex}}$

$0001_{\textbf{bin}} = 1_{\textbf{hex}}$

$0010_{\textbf{bin}} = 2_{\textbf{hex}}$

$0011_{\textbf{bin}} = 3_{\textbf{hex}}$

…

$1111_{\textbf{bin}} = F_{\textbf{hex}}$

Keep shifting and adding 1 from the rightmost position to get all hex symbols, e.g.
0011 -> 0100 -> 0101 -> 0110 -> 0111 -> 1000 is equiv to
3 -> 4 -> 5 -> 6 -> 7 -> 8

Boolean Algebra

- algebra to express binary logic
- basic operators: **OR, AND, NOT**
- OR operators: ∨
  - A ∨ B
- AND operators: ∧
  - A ∧ B
- NOT operators: ¬ -
  - ¬ A $\bar{A}$

AND

Truth Table

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

- Logic Gates: Digital systems are constructed using logic gates that implement functions like AND, OR, NOT.
  - Logic gates are implemented using electronic circuits

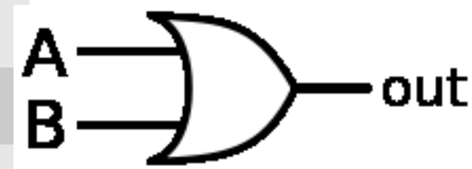
Truth Tables and Gates

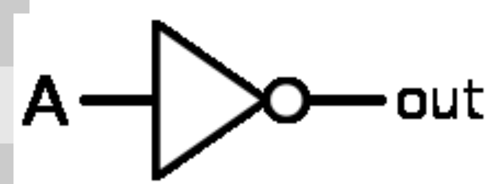
OR	X	Y	Result
X ∨ Y	0	0	0
X + Y	0	1	1
	1	0	1
	1	1	1

AND	X	Y	Result
X ∧ Y	0	0	0
X * Y	0	1	0
	1	0	0
	1	1	1

NOT	X	Result
¬ X	0	1
	1	0

Logic Gate Symbols





Other Rules

Let A be a logic result, i.e. A is in [True, False]

- Identities

$A \vee 0 = A$

$A \wedge 1 = A$

$A \vee A = A$

$A \wedge A = A$

- Involution

$\neg(\neg A) = A$

- Annihilators

$A \vee 1 = 1$

In an OR stmt, if at least one component is 1, the entire stmt results in 1

$A \wedge 0 = 0$

In an AND stmt, if at least one component is 0, the entire stmt results in 0

- Complements

$A \vee \neg A = 1$

One of [A, not A] is 1

$A \wedge \neg A = 0$

One of [A, not A] is 0

Other Rules - 2

- Commutative Law

$A + B = B + A$ *and* $A * B = B * A$

$A \vee B = B \vee A$ *and* $A \wedge B = B \wedge A$

- Associative Law

$A + (B + C) = (A + B) + C$ *and*
$A * (B * C) = (A * B) * C$

$A \vee (B \vee C) = (A \vee B) \vee C$ *and*
$A \wedge (B \wedge C) = (A \wedge B) \wedge C$

Other Rules - 3

- Distributive Law

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

$A \wedge (B \vee C) = A \wedge B \vee A \wedge C$

- De Morgan's Law

$\neg (A \vee B) = \neg A \wedge \neg B$

$\neg (A \wedge B) = \neg A \vee \neg B$

EXCLUSIVE OR

ONLY ONE of X,Y can be true for the statement to be true

- XOR

$X \oplus Y$

X	Y	Result
0	0	0
0	1	1
1	0	1
1	1	0

EXCLUSIVE OR

- XOR

$X \oplus Y$

X	Y	Result
0	0	0
0	1	1
1	0	1
1	1	0

$$X \oplus Y = (\neg X \land Y) \lor (X \land \neg Y)$$

$$X \oplus Y = \neg(X \land Y) \land (X \lor Y)$$

$$X \oplus Y = (\neg X \lor \neg Y) \land (X \lor Y)$$

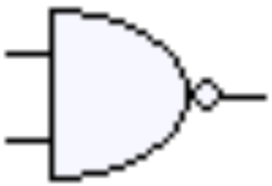
NOT AND

A | B = not(A AND B)

- NAND

X | Y

X	Y	Result
0	0	1
0	1	1
1	0	1
1	1	0



Whereas AND only produces FALSE unless both X,Y true, NAND only produces FALSE iff both X,Y true

- NAND is *functionally complete* (as is NOR)

$\neg X = X \mid X$

FC set of operators: any Boolean expression can be re-expressed by some combination of the operators in the set

$X \wedge Y = (X \mid Y) \mid (X \mid Y)$

= NOT(NOT(X AND Y) AND NOT(X AND Y))
= (X AND Y) OR (X AND Y) by DeMorgan's Law
= (X AND Y)

$X \vee Y = (X \mid X) \mid (Y \mid Y)$

= NOT(NOT(X AND X) AND NOT(Y AND Y))
= (X AND X) OR (Y AND Y) by DeMorgan's Law
= X OR Y

Bytes

1 byte (B)	8 bit
1 kilobyte (K/Kb)	2^{10} byte = 1024 byte
1 megabyte (M/Mb)	2^{20} byte = 1024 Kb
1 gigabyte (G/Gb)	2^{30} byte = 1024 Mb
1 terabyte (T/Tb)	2^{40} byte = 1024 Gb
1 petabyte (P/Pb)	2^{50} byte = 1024 Tb
1 exabyte (E/Eb)	2^{60} byte = 1024 Pb
1 zettabyte (Z/Zb)	2^{70} byte = 1024 Eb
1 yottabyte (Y/Yb)	2^{80} byte = 1024 Zb

Binary Addition

- textbook procedure
- add digits right to left (least significant bit to most significant bit)
  - include carry-over

Carry-over: similar to how we carry-over the 1 in decimal addition when the sum >= 10, we carry over the 1 in binary addition when the sum >= 2

e.g. 011 + 010 = 101

2-Bit Addition

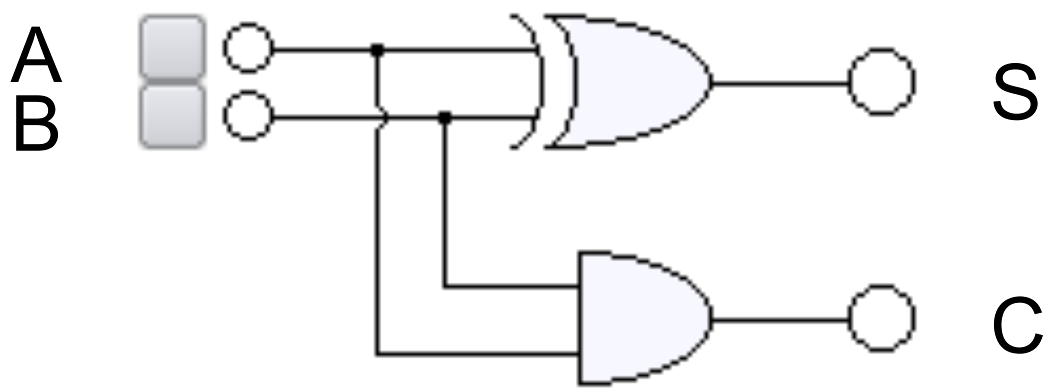
- Truth Table

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- Sum = _____
- Carry = _____

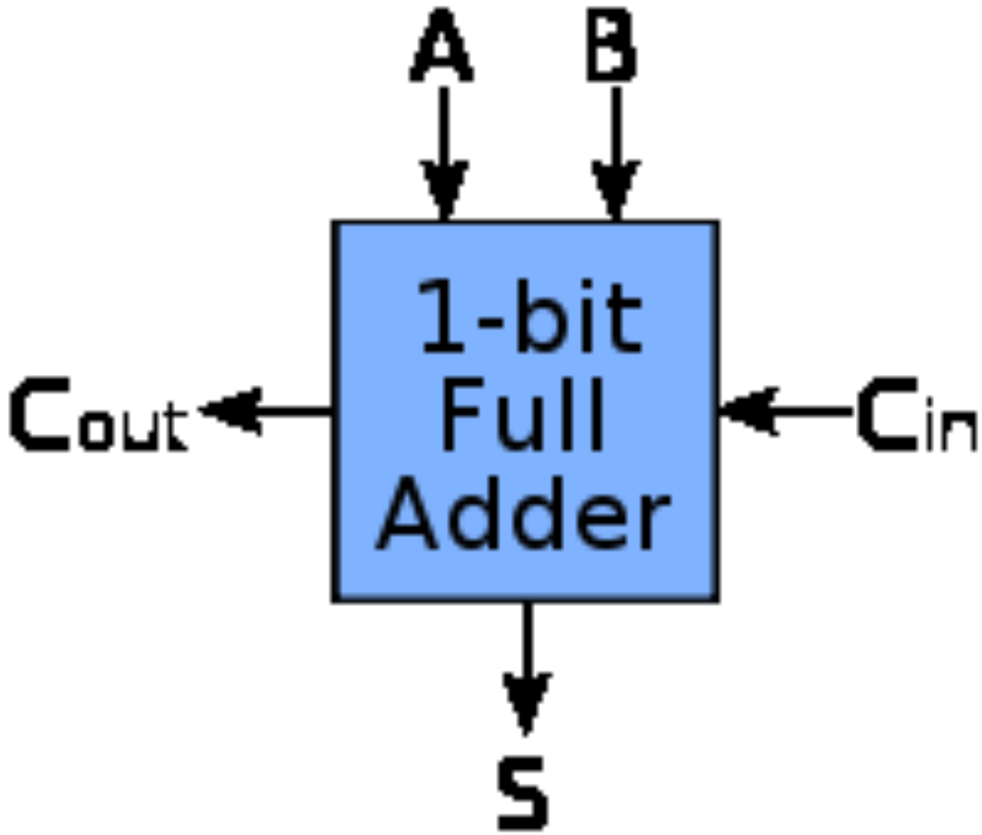
Half Adder

- combine XOR and AND gate



- **S**um and **C**arry
- Carry-in?

Full Adder



2-Bit Addition with Carry

- multiple bits: need to include carry
- Truth Table

A	B	$Carry_{in}$	Sum	$Carry_{out}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

2-Bit Addition with Carry

- Sum: 1 or 3 bits set

$A \oplus B \oplus C_{in}$

Carryout: if at least 2 of A,B,C is 1, the carryout will always be 1 (as the sum >= 2)

- C_{out}: 2 or 3 bits set

Intermediate Goal: express (A v B) in terms of XOR gates
Using truth tables, we see that A v B = (A ^ B) v (A XOR B)

$(A \wedge B) \vee (A \wedge C_{in}) \vee (B \wedge C_{in})$

$(A \wedge B) \vee ((A \vee B) \wedge C_{in})$

$(A \wedge B) \vee ((A \oplus B) \wedge C_{in})$

= (A*B) + (A*C_in) + (B*C_in)
= (A*B) + ((A + B)*C_in)
= (A*B) + (((A * B) + (A XOR B))*C_in)
= (A*B) + (A * B)* C_in + (A XOR B)*C_in
= (A*B)(1 + C_in) + (A XOR B)*C_in
= (A*B) + (A XOR B)*C_in

Full Adder

- comprised of two half adders and OR